

PWNING OWASP WASP JUICE SHOP



BJÖRN KIMMINICH

Table of Contents

Introduction	1.1
Why OWASP Juice Shop exists	1.2
Architecture overview	1.3
Part I - Hacking preparations	2.1
Hacking exercise rules	2.1.1
Walking the "happy path"	2.1.2
Part II - Challenge hunting	3.1
Finding the Score Board	3.1.1
Injection	3.1.2
Broken Authentication	3.1.3
Forgotten Content	3.1.4
Roll your own Security	3.1.5
Sensitive Data Exposure	3.1.6
XML External Entities (XXE)	3.1.7
Improper Input Validation	3.1.8
Broken Access Control	3.1.9
Security Misconfiguration	3.1.10
Cross Site Scripting (XSS)	3.1.11
Insecure Deserialization	3.1.12
Vulnerable Components	3.1.13
Security through Obscurity	3.1.14
Part III - Getting involved	4.1
Provide feedback	4.1.1
Help with translation	4.1.2
Donations	4.1.3
About this book	5.1

Pwning OWASP Juice Shop

Written by [Björn Kimminich](#)



This is the official companion guide to the **OWASP Juice Shop** application. Being a web application with a vast number of intended security vulnerabilities, the OWASP Juice Shop is supposed to be the opposite of a *best practice* or *template application* for web developers: It is an awareness, training, demonstration and exercise tool for security risks in modern web applications. The OWASP Juice Shop is an open-source project hosted by the non-profit [Open Web Application Security Project \(OWASP\)](#) and is developed and maintained by volunteers. The content of this book was written for v7.0.0 of OWASP Juice Shop.

The book is divided into three parts:

Part I - Hacking preparations

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

Part II - Challenge hunting

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

Part III - Getting involved

Part three shows up various ways to contribute to the OWASP Juice Shop open source project.

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



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

Why the Juice Shop exists

To the unsuspecting user the Juice Shop just looks like a small online shop which sells - *surprise!* - fruit & vegetable juice and associated products. Except for the entirely overrated payment and delivery aspect of the e-commerce business, the Juice Shop is fully functional. But this is just the tip of the iceberg. The Juice Shop contains 59 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.

LAB medium level projects



Why the name "Juice Shop"?

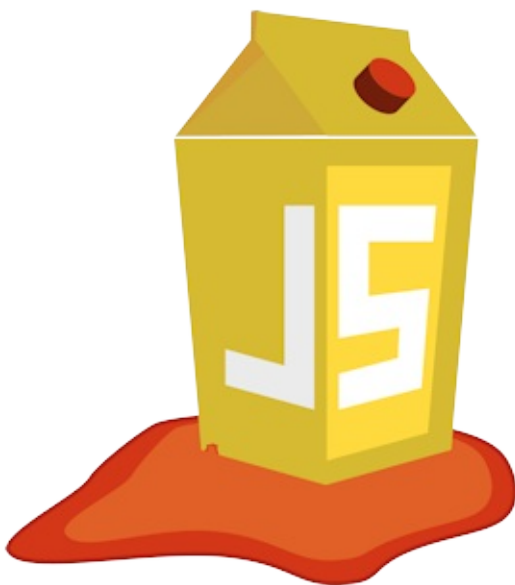
In German there is a dedicated word for *dump*, i.e. a store that sells lousy wares and does not exactly have customer satisfaction as a priority: *Saftladen*. Reverse-translating this separately as *Saft* and *Laden* yields *juice* and *shop* in English. That is where the project name comes from. The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

Why the logo?

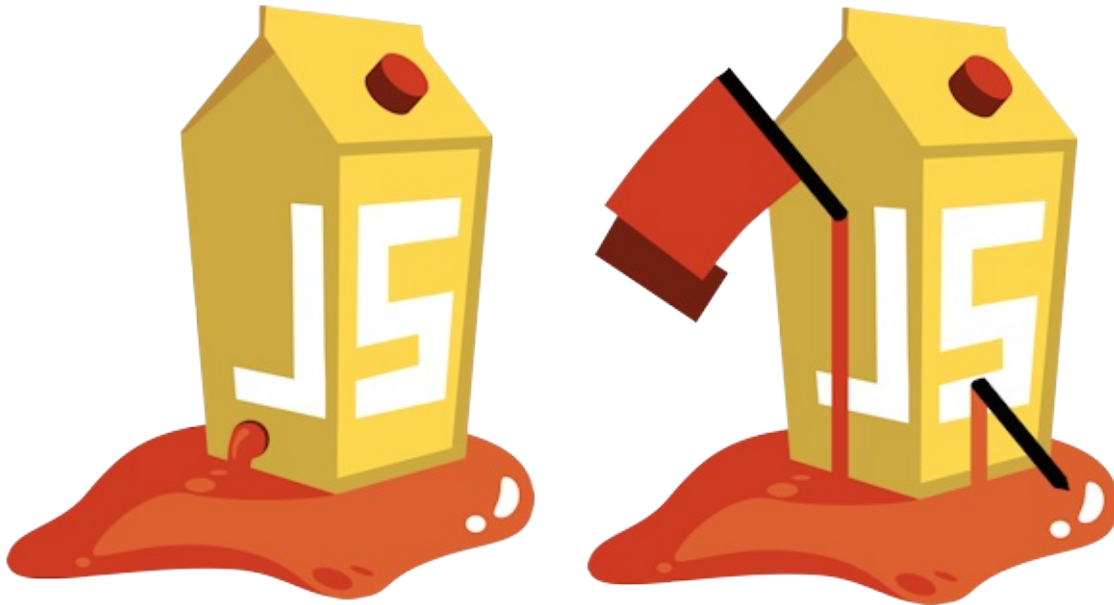
Other than the name, the Juice Shop logo was designed explicitly with *JavaScript* in mind:



The 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 \(VWAD\)](https://www.owasp.org) 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

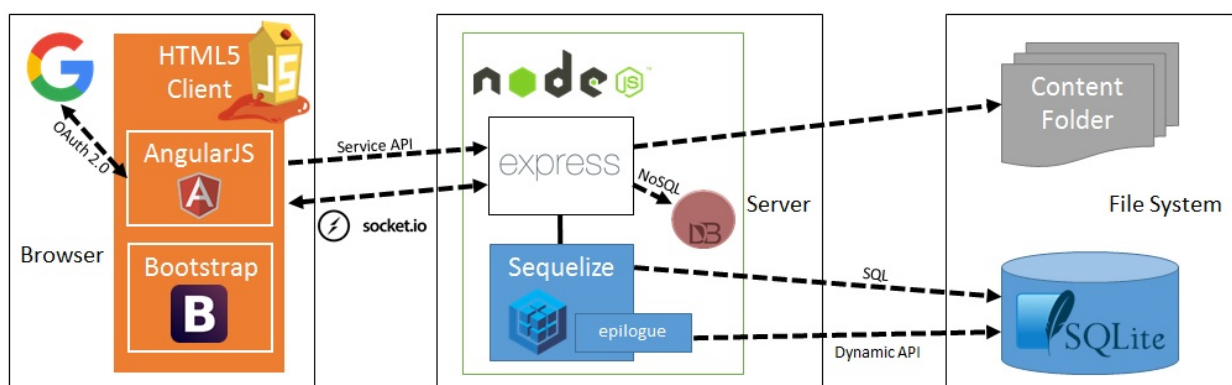
The OWASP Juice Shop is a pure web application implemented in JavaScript. In the frontend the popular [AngularJS](#) framework is used to create a so-called *Single Page Application*. The user interface layout is provided by Twitter's [Bootstrap](#) framework - which works nicely in combination with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

Hacking exercise rules

✓ Recommended hacking tools

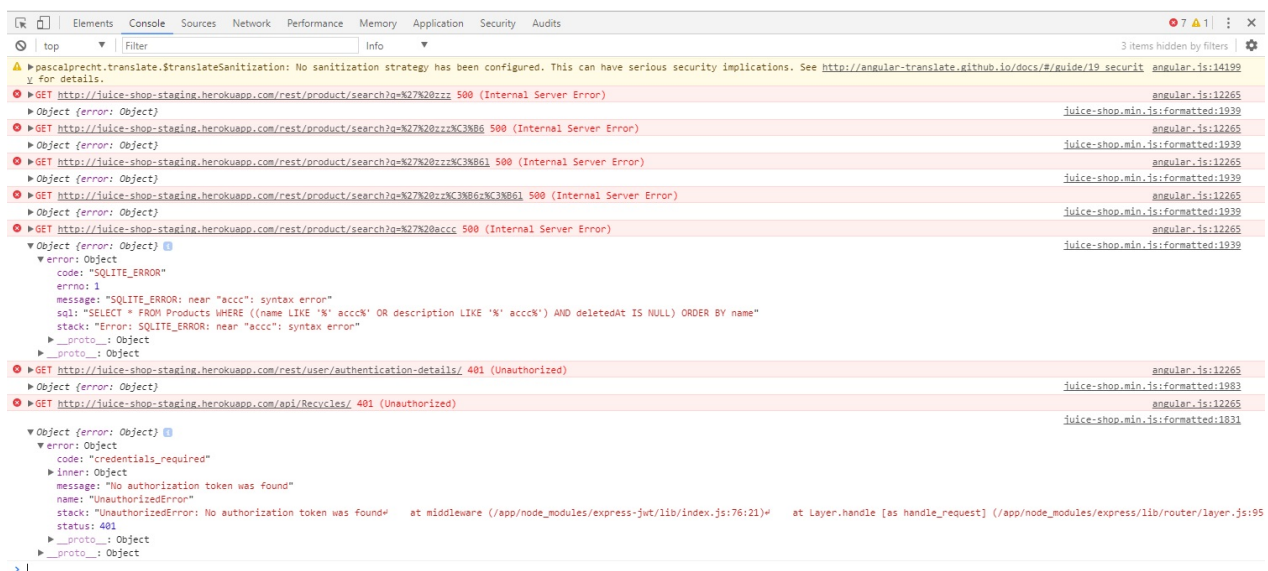
Browser

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.

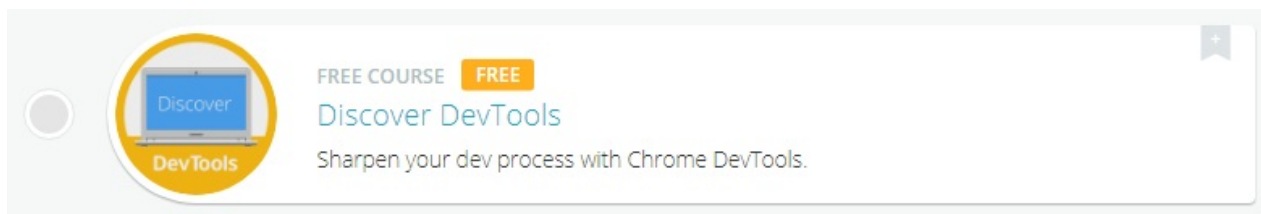
When hacking a web application that relies heavily on JavaScript, **it is essential to your success to monitor the *JavaScript Console* permanently!** It might leak valuable information to you through error or debugging logs!



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.

Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Secure	SameSite
continueCode	qIO9pJ8bzNIYDxkMgy3XVwLQAjvHquyh9Aqm4E...	juice-shop-s...	/	2017-07-31...	72			
io	2VVMtNnumLZEMmrFAAAB	juice-shop-s...	/	Session	22	✓		
token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dX...	juice-shop-s...	/	Session	398			

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

🔑 TODO

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.

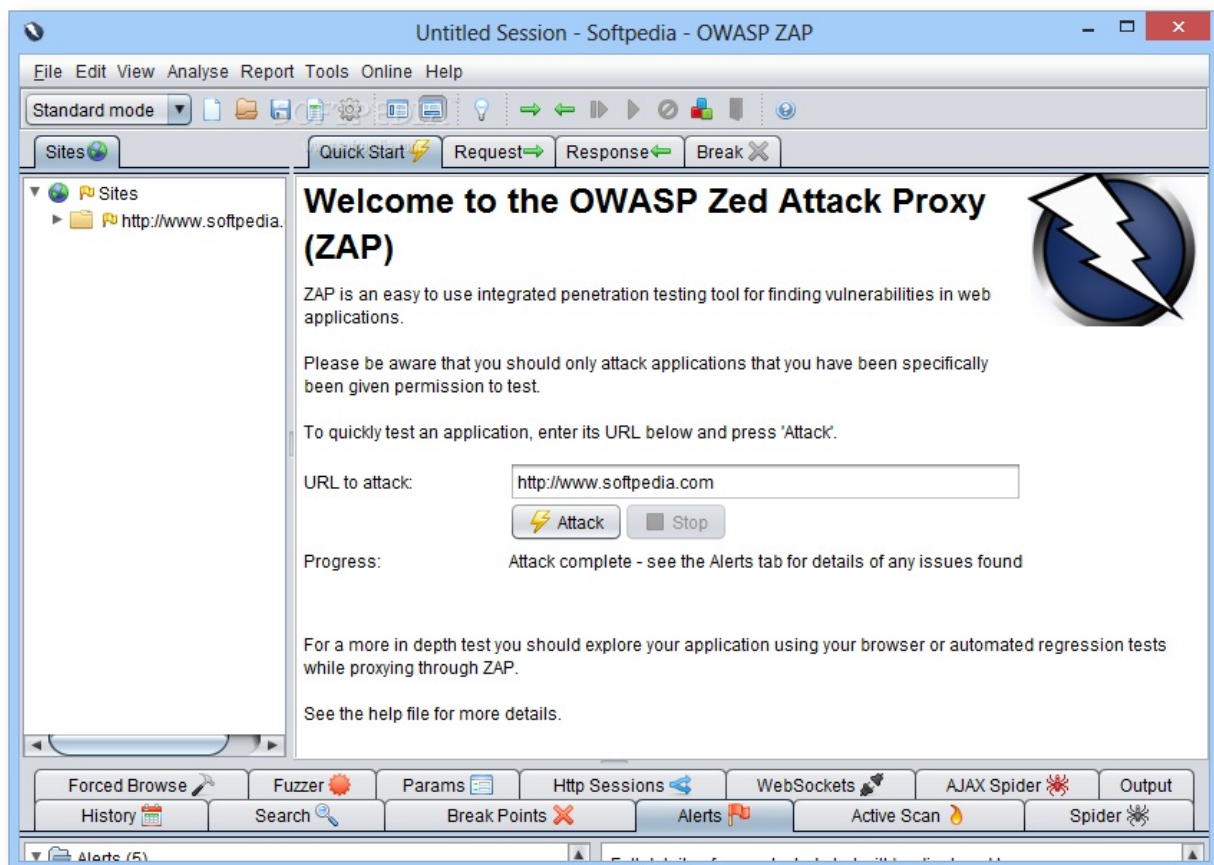
In the following sections you find some recommended pentesting tools in case you want to try one. Please be aware that the tools are not trivial to learn - let alone master. Trying to learn about the web application security basics *and* hacking tools *at the same time* is unlikely to get you very far 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.¹



Pentesting Linux distributions

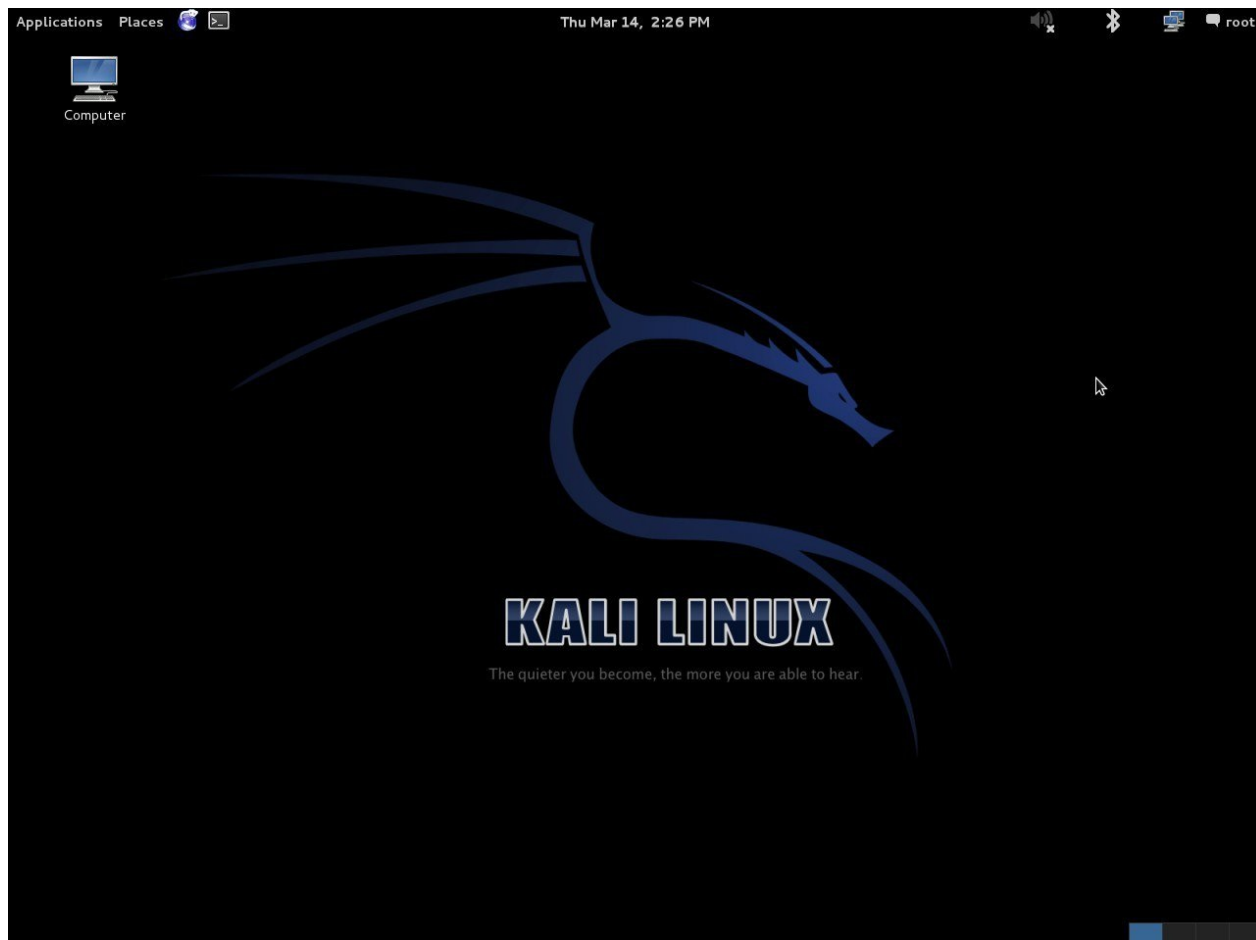
Instead of installing a tool such as ZAP on your computer, why not take it, add *several hundred* of other offensive security tools and put them all into a ready-to-use Linux distribution? Entering [Kali Linux](#) and similar toolboxes:

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.²

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.³

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

You have been warned.

Source code

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

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

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

GitHub repository

While stated earlier that "the Internet" is fine as a helpful resource, consider the GitHub repository 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

🔑 TODO

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://github.com/zaproxy/zap-core-help/wiki> ↩

². <http://docs.kali.org/introduction/what-is-kali-linux> ↩

³. <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_\(OTG-INFO-007\)](https://www.owasp.org/index.php/Map_execution_paths_through_application_(OTG-INFO-007)) ↩

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

Part II - Challenge hunting

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

















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

The challenge hints found in this release of the companion guide are compatible with v7.0.0 of OWASP Juice Shop.

Trivial Challenges (★)

Challenge	Description	Hints	Solution
Admin Section	Access the administration section of the store.	💡	👤
Confidential 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 1	Perform a reflected XSS attack with <code><script>alert("XSS")</script></code> .	💡	👤
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.		
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 <i>the truthful answer</i> 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 <code><script>alert("XSS")</script></code> bypassing a client-side security mechanism.	💡	👤
XSS Tier 3	Perform a persisted XSS attack with <code><script>alert("XSS")</script></code> without using the frontend application at all.	💡	👤
XXE Tier 1	Retrieve the content of <code>c:\Windows\system.ini</code> or <code>/etc/passwd</code> from the server.	💡	👤













Hard Challenges (★★★★★)

Challenge	Description	Hints	Solution
CSRF	Change Bender's password into <i>slurmCl4ssic</i> without using SQL Injection.	💡	👤
Easter Egg Tier 1	Find the hidden easter egg.	💡	👤
Easter Egg Tier 2	Apply some advanced cryptanalysis to find <i>the real</i> 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 <i>the truthful answer</i> to his security question.	💡	👤
Typosquatting Tier 1	Inform the shop about a <i>typosquatting</i> 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 <code><script>alert("XSS")</script></code> 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 <i>jwt3d@juice-sh.op</i> .	💡	👤
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 <i>the truthful answer</i> to his security question.	💡	👤
Reset Morty's Password	Reset Morty's password via the Forgot Password mechanism with <i>his obfuscated answer</i> to his security question.	💡	👤
Retrieve Blueprint	Deprive the shop of earnings by downloading the blueprint for one of its products	💡	👤
Typosquatting Tier 2	Inform the shop about a more literal instance of <i>typosquatting</i> 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 <i>rsa_lord@juice-sh.op</i> .		
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.		

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 at 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 <i>the truthful answer</i> to his security question.	☆☆☆☆
Change Bender's password into <i>slurmCl4ssic</i> 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 <i>the truthful answer</i> 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 <i>the truthful answer</i> to his security question.	☆☆☆☆☆ ☆

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 slurmCI4ssic without using SQL Injection

This challenge can only be solved by changing the password of user Bender into *slurmCI4ssic*. 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

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.

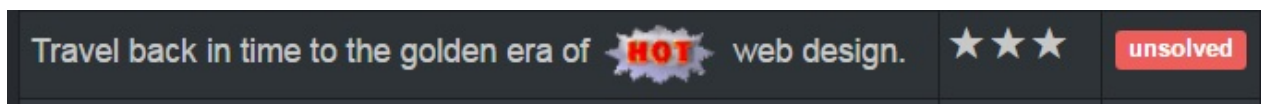
Hints

- 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



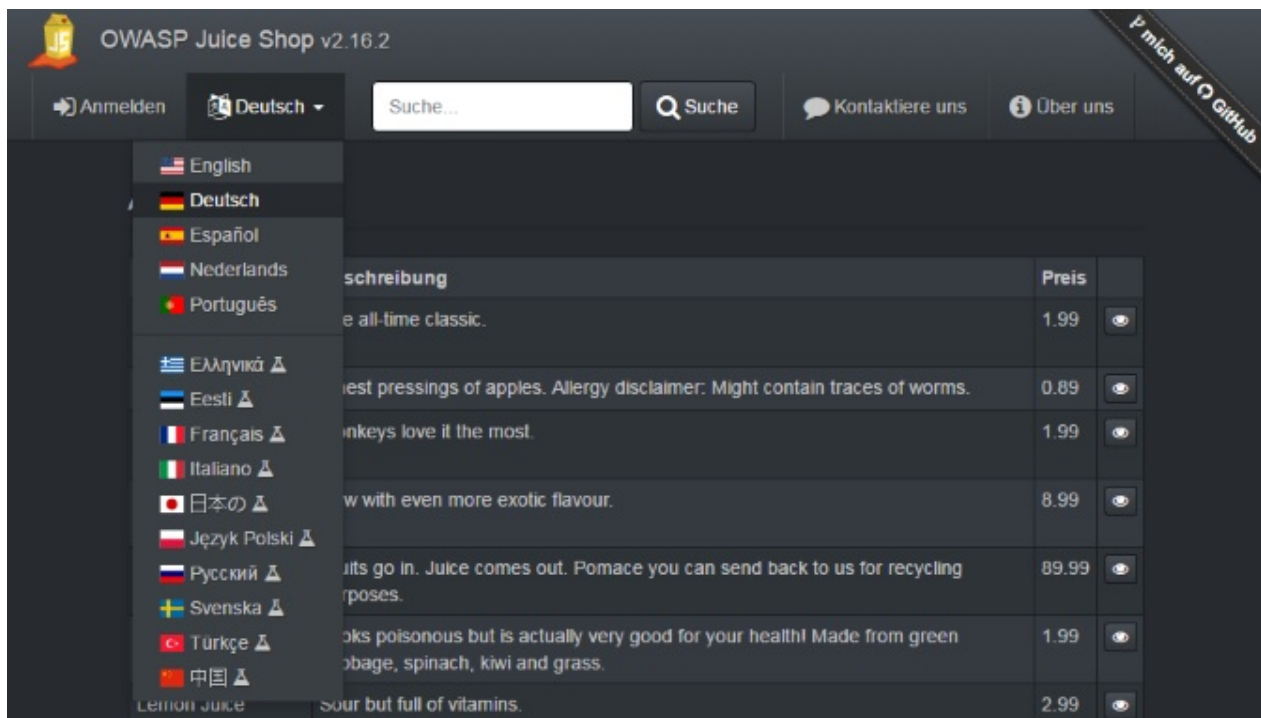
- The mentioned *golden era* lasted from 1994 to 2009.
- You can solve this challenge by requesting a specific file

*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 too 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

i 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.

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

Roll your own Security

Challenges covered in this chapter

Challenge	Difficulty
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.	★★★★★

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 the 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

🔑 TODO

Hints

🔑 TODO

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

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.



Rapper Who Is Very Concerned With Password Security

1,255,120 views

👍 28K 💬 701 ➦ SHARE ≡ ...

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

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

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

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

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

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

A good understanding of crypto is required to be able to discern between solid products and snake oil. The inherent complexity of crypto makes it easy to fall for fantastic claims from vendors about their product. Typically, these are “a breakthrough in cryptography” or “unbreakable” or provide “military grade” security. If a vendor says “trust us, we have had experts look at this,” chances are they weren’t experts¹

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 <code>c:\Windows\system.ini</code> or <code>/etc/passwd</code> 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 basically gives away the solution for free.

Give the server something to chew on for quite a while

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

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](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: `More...`
- Expected link tag in the description: `More...`

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 <i>his obfuscated answer</i> 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 <code><script>alert("XSS")</script></code> .	★
Perform a persisted XSS attack with <code><script>alert("XSS")</script></code> bypassing a client-side security mechanism.	★★★
Perform a persisted XSS attack with <code><script>alert("XSS")</script></code> without using the frontend application at all.	★★★
Perform a persisted XSS attack with <code><script>alert("XSS")</script></code> 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 persisted XSS attack bypassing a client-side security mechanism

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

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

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.

Name	Status	Type	Initiator
<input type="checkbox"/> search?q=undefined	304	xhr	angular.js:12011
<input type="checkbox"/> 1?d=Mon%20Nov%2007%202016	200	xhr	angular.js:12011
<input type="checkbox"/> whoami	304	xhr	angular.js:12011
<input type="checkbox"/> Feedbacks/	200	xhr	angular.js:12011
<input type="checkbox"/> Feedbacks/	200	xhr	angular.js:12011
<input type="checkbox"/> search?q=undefined	304	xhr	angular.js:12011
<input type="checkbox"/> search?q=apple	200	xhr	angular.js:12011
<input type="checkbox"/> 17?d=Mon%20Nov%2007%202016	200	xhr	angular.js:12011
<input type="checkbox"/> login	404	xhr	angular.js:12011
<input type="checkbox"/> login	200	xhr	angular.js:12011
<input type="checkbox"/> search?q=undefined	304	xhr	angular.js:12011

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 be 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_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)) ↩

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

3. 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 make the server busy for all eternity.
- The challenge will be solved if you manage to trigger the protection of the application against such DoS attacks.
- 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

🔑 TODO

Hints

🔑 TODO

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 ↩

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 <i>jwt3d@juice-sh.op</i> .	★★★★★ ★
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 <i>rsa_lord@juice-sh.op</i> .	★★★★★ ★★

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: `examplpe.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 typosquatting 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 <i>the real</i> easter egg.	☆☆☆☆

Learn about the Token Sale before its official announcement

🔑 TODO

Hints

🔑 TODO

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.

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

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=PLV9O4rlovHhO1y8_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!

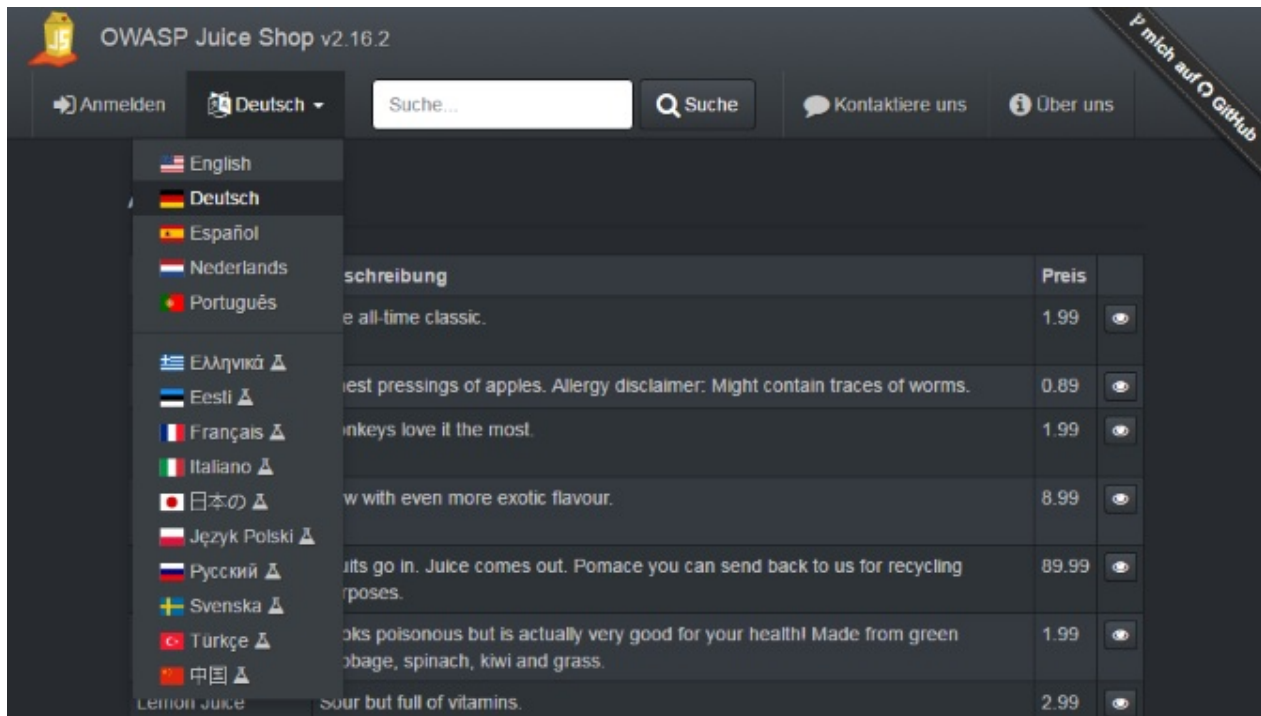
Feedback Channels

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!

Helping with translations

The user interface of OWASP Juice Shop is fully translated into several languages. For many other 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](https://crowdin.com/project/owasp-juice-shop) 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

bkimminich's projects / OWASP Juice Shop

OWASP Juice Shop

OWASP Juice Shop is an intentionally insecure webapp for security trainings written entirely in Javascript which encompasses the entire OWASP Top Ten and other severe security flaws.

Translations Activity Discussions

You Preferred:



German
approved: 100%

Needs Translation:



Chinese Simplified
translated: 0%



Dutch
translated: 93%



Estonian
translated: 11%



Finnish
translated: 0%



French
translated: 4%



Greek
translated: 0%



Italian
translated: 2%



Japanese
translated: 0%



Klingon
translated: 16%



Latvian
translated: 0%



Lithuanian
translated: 0%



Polish
translated: 57%



Portuguese
translated: 93%



Russian
translated: 43%



Spanish
translated: 93%



Turkish
translated: 0%

Completed:



German
approved: 100%



Swedish
approved: 100%

Settings
Edit Project

Download
Translations in a ZIP Archive

Source language:

 English

Owner:

 Björn Kimmich
bkimminich
Contact

10 users participate in this project

Created: 3 months ago
Last Activity: 5 days ago
Last build: 5 days ago

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 `110n_develop` Git branch to synchronize translations into the `app/i18n/??/json` language files where `??` is a language code (e.g. `en` or `de`).

Adding another language

If you do not find the language you would like to provide a translation for in the list, please contact the OWASP Juice Shop [project leader](#) or [raise an issue on GitHub](#) asking for the missing language. It will be added asap!

Translating directly via GitHub PR

1. Fork the repository <https://github.com/bkimminich/juice-shop>
2. Translate the labels in the desired language- `.json` file in `/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.



The screenshot shows the 'OWASP Foundation Donation' registration page on RegOnline. The page header includes the RegOnline logo and a 'Host Your Own Event' button. The main content area features the OWASP logo and the text 'The Open Web Application Security Project'. Below this, it says 'OWASP Foundation Donation'. There is a section for contact information: 'United States', 'Phone: 301-275-9403', and a link to 'Email Us'. A link for 'View Your Existing Registration' is also present. The 'Start Your Registration' section contains a form with two fields: 'Email Address' (with the value 'bjoern.kimminich@owasp.org') and 'Donation Type' (a dropdown menu with options 'Chapter Supporter', 'Project Supporter', and 'Foundation Donor'). The 'Project Supporter' option is currently selected.

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*.

Agenda

- ☐ Platinum Project Supporter
Price: \$250.00
- ☐ Gold Project Supporter
Price: \$100.00
- ☐ Silver Project Supporter
Price: \$20.00
- ☒ Project Supporter - Other


\$ 10 Project Donation Amount

OWASP Juice Shop Project ★ Which Project would you like to support?

Total: 00

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.


[Personal Info](#) [Agenda](#) [Checkout](#) **[Confirmation](#)**




Your registration is complete.

A confirmation email has been sent to bjoern.kimminich@owasp.org.


Now, invite your friends and co-workers!




Invite Your Friends



Send an Email




Tweet About It



Tell Your Network

*** FORWARD THIS TO THE CHAPTER/PROJECT LEADER AS PROOF OF PAYMENT ***

Personal Info


Registration ID: 

Registrant: Bjoern Kimminich
Germany


Registration Date: 11/19/2017 11:32 PM

Donation: Project Supporter


Status: Confirmed


Work Phone: +49 

Email: bjoern.kimminich@owasp.org




Actions

 [Mobile Event Guide](#)

 [Print Your Registration](#)

Documents

 [Receipt](#)

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.

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.

