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.	1 of 5

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

Most applications contain URLs which are not supposed to be publicly accessible. A properly implemented authorization model would ensure that only users *with appropriate permission* can access such a URL. If an application instead relies on the fact that the URL 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]

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

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

Information Leakage

> 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]

Challenges covered in this chapter

Challenge	Difficulty
Provoke an error that is not very gracefully handled.	1 of 5

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.

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?

> 1. https://www.owasp.org/index.php/Top_10_2007-Information_Leakage ↩

SQL 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
Log in with the administrator's user account.	2 of 5
Order the Christmas special offer of 2014.	2 of 5
Retrieve a list of all user credentials via SQL Injection	3 of 5
Log in with Jim's user account.	3 of 5
Log in with Bender's user account.	3 of 5

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.

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.

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*.

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

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 of course try to attack that instead of SQL Injection.

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

- You should try one of the approaches you used on Jim.

- Bender's password hash might not help you very much.

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

Privilege escalation

> 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.	1 of 5
Get rid of all 5-star customer feedback.	1 of 5
Change the href of the link within the O-Saft product description into http://kimminich.de.	3 of 5
Access someone else's basket.	2 of 5
Post some feedback in another users name.	3 of 5

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

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!

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

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

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

Hints

- *Theoretically* there are three possible ways to beat this challenge:
  - Finding an administrative functionality in the web application that lets you change product data
  - Looking for possible holes in the RESTful API that would allow you to update a product
  - Attempting an SQL Injection attack that sneaks in an `UPDATE` statement on product data
- *In practice* two of these three ways should turn out to be dead ends

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.

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.

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

Forgotten content

The challenges in this chapter are all about files that were either forgotten, accidentally misplaced or have been added as a joke by the development team.

Challenges covered in this chapter

Challenge	Difficulty
Access a confidential document.	1 of 5
Access a salesman's forgotten backup file.	2 of 5
Access a developer's forgotten backup file.	3 of 5
Find the hidden easter egg.	3 of 5
Travel back in time to the golden era of web design.	3 of 5
Deprive the shop of earnings by downloading the blueprint for one of its products.	3 of 5
Retrieve the language file that never made it into production.	4 of 5

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*.

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 pull this trick.

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.

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

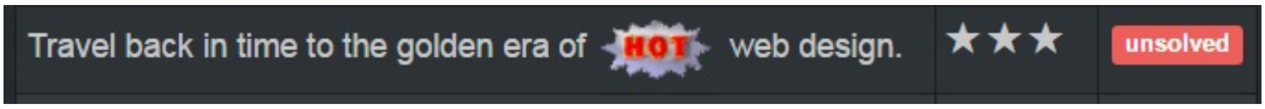
- If you solved one of the three file access challenges above, you already know where the easter egg is located
- Simply reuse one of the tricks that already worked for the files above

*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.*

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



Travel back in time to the golden era of web design. ★★★ unsolved

- 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.*

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?

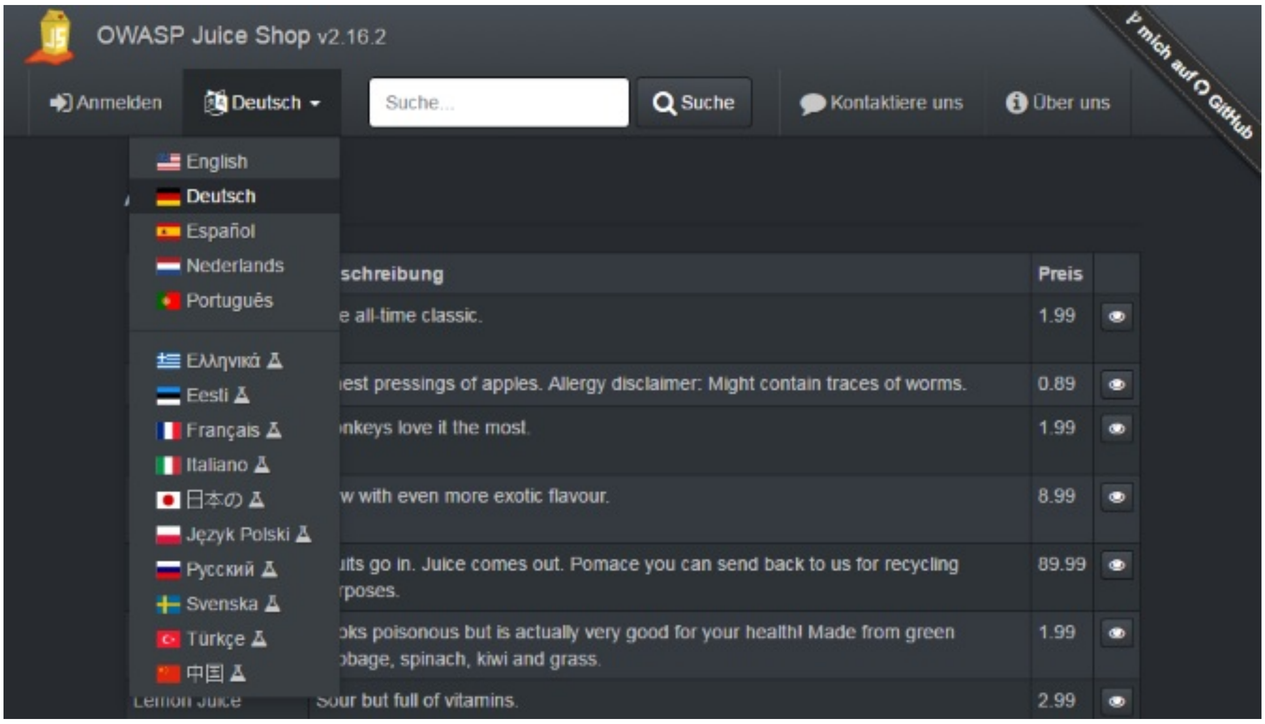
Hints

- 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

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.[2]

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.


OWASP Juice Shop v2.16.2
Anmelden
Deutsch
Suche...
Suche
Kontaktiere uns
Über uns
mich auf GitHub
English
Deutsch
Español
Nederlands
Português
Ελληνικά
Eesti
Français
Italiano
日本の
Język Polski
Русский
Svenska
Türkçe
中国
schreibung
Preis
e all-time classic.
1.99
est pressings of apples. Allergy disclaimer: Might contain traces of worms.
0.89
nkeys love it the most.
1.99
w with even more exotic flavour.
8.99
its go in. Juice comes out. Pomace you can send back to us for recycling
poses.
89.99
oks poisonous but is actually very good for your health! Made from green
bbage, spinach, kiwi and grass.
1.99
Lemon Juice
Sour but full of vitamins.
2.99

Hints

- First you should find out how the languages are technically changed in the user interface.
- You can then choose between three viable ways to beat this challenge:
  - Trust in your luck and *guess* what language is the extra one.
  - *Apply brute force* (and don't give up to quickly) to find it.
  - *Investigate externally* what languages are actually available.

> 1. https://en.wikipedia.org/wiki/Easter_egg_(media) ↩

> 2. https://www.owasp.org/index.php/OWASP_2014_Project_Handbook#tab=Project_Requirements ↩

Cross Site Scripting (XSS)

> Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.
>
> An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.[1]

Challenges covered in this chapter

Challenge	Difficulty
Perform a reflected XSS attack with `<script>alert("XSS1")</script>`.	1 of 5
Perform a persisted XSS attack with `<script>alert("XSS2")</script>` bypassing a client-side security mechanism.	3 of 5
Perform a persisted XSS attack with `<script>alert("XSS3")</script>` without using the frontend application at all.	3 of 5
Perform a persisted XSS attack with `<script>alert("XSS4")</script>` bypassing a server-side security mechanism.	4 of 5

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:

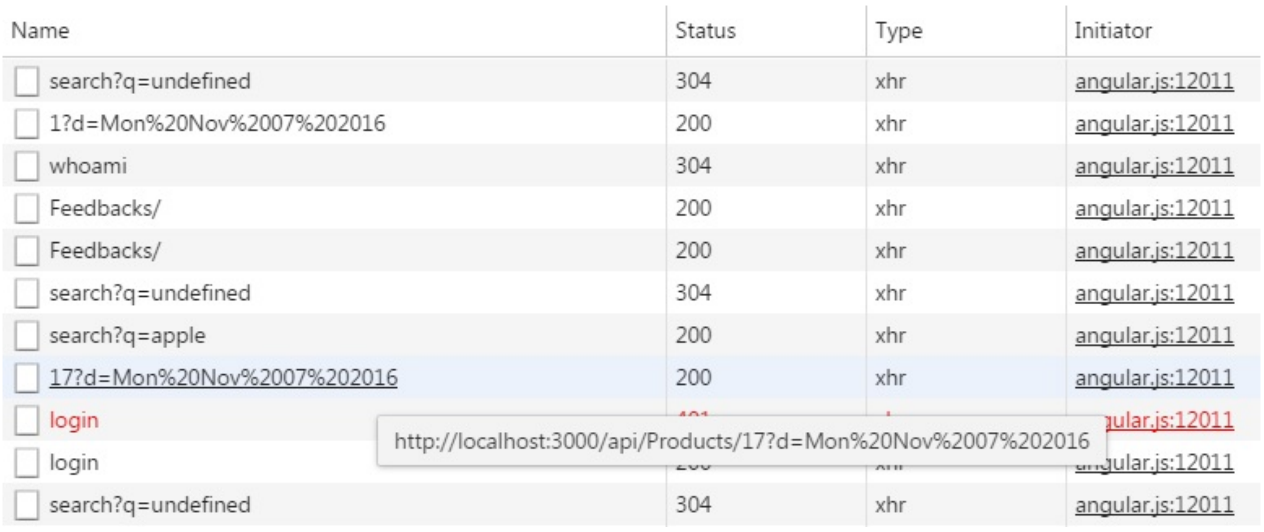
> Be aware that any JavaScript input validation performed on the client can be bypassed by an attacker that disables JavaScript or uses a Web Proxy. Ensure that any input validation performed on the client is also performed on the server.[3]

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
search?q=undefined	304	xhr	angular.js:12011
1?d=Mon%20Nov%2007%202016	200	xhr	angular.js:12011
whoami	304	xhr	angular.js:12011
Feedbacks/	200	xhr	angular.js:12011
Feedbacks/	200	xhr	angular.js:12011
search?q=undefined	304	xhr	angular.js:12011
search?q=apple	200	xhr	angular.js:12011
17?d=Mon%20Nov%2007%202016	200	xhr	angular.js:12011
login	[illegible]	[illegible]	[illegible]
login	[illegible]	[illegible]	[illegible]
search?q=undefined	304	xhr	angular.js:12011

http://localhost:3000/api/Products/17?d=Mon%20Nov%2007%202016

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

Hints

- A matrix of known data entities and their supported HTTP verbs through the API can help you here
- Careless developers might have exposed API methods that the client does not even need

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

This is the hardest XSS challenge, as it cannot by solved by fiddling with the client-side JavaScript or bypassing the client entirely. Whenever there is a server-side validation or input processing involved, you should investigate how it works. Finding out implementation details

- e.g. used libraries, modules or algorithms - should be your priority. If the application does not leak this kind of details, you can still go for a *blind approach* by testing lots and lots of different attack payloads and check the reaction of the application.

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

Hints

- The *Comment* field in the *Contact Us* screen is where you want to put your focus on
- The attack payload `<script>alert("XSS4")</script>` will *not be rejected* by any validator

but *stripped from the comment* before persisting it
- Look for possible dependencies related to input processing in the `package.json.bak` you harvested earlier

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

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

> 3. https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet#Client_Side_vs_Server_Side_Validation ↩

Cross Site Request Forgery (CSRF)

> 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.[1]

Challenges covered in this chapter

Challenge	Difficulty
Change Bender's password into *slurmCl4ssic* without using SQL Injection.	4 of 5

Change Bender's password into slurmCl4ssic without using SQL Injection

This challenge can only be solved by changing the password of user Bender into *slurmCl4ssic*. Using any sort of SQL Injection will *not* solve the challenge, even if the password is successfully changed in the process.

Hints

- The fact that this challenge is in the CSRF category 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.

> 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.[2]

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

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

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 vulnerable library it is using. (Mention the exact library name and version in your comment)	3 of 5
Inform the shop about a typosquatting trick it has become victim of. (Mention the exact name of the culprit)	3 of 5
Inform the shop about a more literal instance of typosquatting it fell for. (Mention the exact name of the culprit)	4 of 5

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.

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

> Typosquatting, also called URL hijacking, a sting site, or a fake URL, is a form of cybersquatting, and possibly brandjacking which relies on mistakes such as typos made by Internet users when inputting a website address into a web browser. Should a user accidentally enter an incorrect website address, they may be led to any URL (including an alternative website owned by a cybersquatter).
>
> The typosquatter's URL will usually be one of four kinds, all similar to the victim site address (e.g. example.com):
>
> - A common misspelling, or foreign language spelling, of the intended site: exemple.com
> - A misspelling based on typos: examlpe.com
> - A differently phrased domain name: examples.com
> - A different top-level domain: example.org
> - An abuse of the Country Code Top-Level Domain (ccTLD): example.cm by using .cm, example.co by using .co, or example.om by using .om. A person leaving out a letter in .com in error could arrive at the fake URL's website.
>
> Once in the typosquatter's site, the user may also be tricked into thinking that they are in fact in the real site, through the use of copied or similar logos, website layouts or content. Spam emails sometimes make use of typosquatting URLs to trick users into visiting malicious sites that look like a given bank's site, for instance.[1]

This challenge is about identifying and reporting (via the http://localhost:3000/#/contact form) a case of typosquatting that successfully sneaked into the Juice Shop. In this case, there is no actual malice or mischief included, as the typosquatter is completely harmless. Just keep in mind that in reality, a case like this could come with negative consequences and would sometimes be even harder to identify.

Hints

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

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

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

Hints

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

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

Cryptographic issues

> Initially confined to the realms of academia and the military, cryptography has become ubiquitous thanks to the Internet. Common every day uses of cryptography include mobile phones, passwords, SSL, smart cards, and DVDs. Cryptography has permeated everyday life, and is heavily used by many web applications.
>
> Cryptography (or crypto) is one of the more advanced topics of information security, and one whose understanding requires the most schooling and experience. It is difficult to get right because there are many approaches to encryption, each with advantages and disadvantages that need to be thoroughly understood by web solution architects and developers. In addition, serious cryptography research is typically based in advanced mathematics and number theory, providing a serious barrier to entry.
>
> The proper and accurate implementation of cryptography is extremely critical to its efficacy. A small mistake in configuration or coding will result in removing a large degree of the protection it affords and rending the crypto implementation useless against serious attacks.
>
> A good understanding of crypto is required to be able to discern between solid products and snake oil. The inherent complexity of crypto makes it easy to fall for fantastic claims from vendors about their product. Typically, these are “a breakthrough in cryptography” or “unbreakable” or provide "military grade" security. If a vendor says "trust us, we have had experts look at this,” chances are they weren't experts[1]

Challenges covered in this chapter

Challenge	Difficulty
Inform the shop about an algorithm or library it should definitely not use the way it does.	2 of 5
Apply some advanced cryptanalysis to find *the real* easter egg.	4 of 5
Forge a coupon code that gives you a discount of at least 80%.	5 of 5
Solve challenge #99. Unfortunately, this challenge does not exist.	5 of 5
Unlock Premium Challenge to access exclusive content.	5 of 5

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*.

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.

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.

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 ↩

Validation flaws

Challenges covered in this chapter

Challenge	Difficulty
Give a devastating zero-star feedback to the store.	1 of 5
Place an order that makes you rich.	3 of 5
Upload a file larger than 100 kB.	3 of 5
Upload a file that has no .pdf extension.	3 of 5

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

Weak security mechanisms

Challenges covered in this chapter

Challenge	Difficulty
Log in with the administrator's user credentials without previously changing them or applying SQL Injection.	2 of 5
Log in with Bjoern's user account without previously changing his password, applying SQL Injection, or hacking his Google account.	3 of 5
Exploit OAuth 2.0 to log in with the Chief Information Security Officer's user account.	4 of 5
Wherever you go, there you are.	4 of 5
Log in with the support team's original user credentials without applying SQL Injection or any other bypass.	5 of 5
Forge an essentially unsigned JWT token that impersonates the (non-existing) user *jwtn3d@juice-sh.op*.	4 of 5
Forge an almost properly RSA-signed JWT token that impersonates the (non-existing) user *rsa_lord@juice-sh.op*.	5 of 5

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*

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.

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.

Wherever you go, there you are

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

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

Hints

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

> White list validation involves defining exactly what *is* authorized, and by definition, everything else is not authorized.[2]

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.

Forge an essentially unsigned JWT token

TODO

Hints

TODO

Forge an almost properly RSA-signed JWT token

TODO

Hints

TODO

> 1. https://tools.ietf.org/html/rfc7519 ↩

> 2. https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet#White_List_Input_Validation ↩

Sensitive data exposure

> 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]

Challenges covered in this chapter

Challenge	Difficulty
Reset Jim's password via the Forgot Password mechanism with the original answer to his security question.	2 of 5
Reset Bender's password via the Forgot Password mechanism with the original answer to his security question.	3 of 5
Reset Bjoern's password via the Forgot Password mechanism with the original answer to his security question.	4 of 5

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.

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

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
- Brute forcing the answer should be next to impossible

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 in social networks
- Brute forcing the answer should be next to impossible

1

> 1. http://goodsecurityquestions.com ↩

NoSQL Injection

> 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.[1]

Challenges covered in this chapter

Challenge	Difficulty
Let the server sleep for some time. (It has done more than enough hard work for you)	3 of 5
Update multiple product reviews at the same time.	3 of 5

Let the server sleep for some time

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.[2]

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

🔧 **TODO**

Hints

- Take a close look on how the equivalent of UPDATE-statements in MongoDB work.
- 🔧 **TODO**

> 1. https://www.owasp.org/index.php/Testing_for_NoSQL_injection ↩

> 2. https://www.us-cert.gov/ncas/tips/ST04-015 ↩

Part III - Getting involved

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

Social Media Channels

Channel	Link
GitHub	https://github.com/bkimminich/juice-shop
Twitter	https://twitter.com/owasp_juiceshop
Facebook	https://www.facebook.com/owasp.juiceshop
Open Hub	https://www.openhub.net/p/juice-shop
Community Chat	https://gitter.im/bkimminich/juice-shop
OWASP Slack Channel	https://owasp.slack.com/messages/project-juiceshop
Project Mailing List	owasp_juice_shop_project@lists.owasp.org
Youtube Playlist	https://www.youtube.com/playlist?list=PLV9O4rIovHhO1y8_78GZfMbH6oznyx2g2

Provide feedback

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

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

Feedback Channels

Channel	Link
GitHub Issues	https://github.com/bkimminich/juice-shop/issues
Tweet via @owasp_juiceshop	https://twitter.com/intent/tweet?via=owasp_juiceshop
Community Chat	https://gitter.im/bkimminich/juice-shop
OWASP Slack Channel	https://owasp.slack.com/messages/project-juiceshop
Project Mailing List	owasp_juice_shop_project@lists.owasp.org

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

Contribute to development

If you would like to contribute to OWASP Juice Shop but need some idea what task to address, the best place to look is in the GitHub issue lists at https://github.com/bkimminich/juice-shop/issues.

- Issues labelled with **help wanted** indicate tasks where the project team would very much appreciate help from the community
- Issues labelled with **good first issue** indicate tasks that are isolated and not too hard to implement, so they are well-suited for new contributors

The following sections describe in detail the most important rules and processes when contributing to the OWASP Juice Shop project.

Tips for newcomers

If you are new to application development - particularly with AngularJS and Express.js - it is recommended to read the Codebase 101 to get an overview what belongs where. It will lower the entry barrier for you significantly.

Version control

The project uses `git` as its version control system and GitHub as the central server and collaboration platform. OWASP Juice Shop resides in the following repository:

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

Branching model

OWASP Juice Shop is maintained in a simplified Gitflow fashion, where all active development happens on the `develop` branch while `master` is used to deploy stable versions to the Heroku demo instance and later create tagged releases from.

Feature branches are only used for long-term tasks that could jeopardize regular releases from `develop` in the meantime. Likewise prototypes and experiments must be developed on an individual branch or a distinct fork of the entire project.

Versioning

Any release from `master` is tagged with a unique version in the format `vMAJOR.MINOR.PATCH`, for example `v1.3.0` or `v4.1.2`.

> Given a version number `MAJOR.MINOR.PATCH`, increment the:
>
> 1. `MAJOR` version when you make incompatible API changes,
> 2. `MINOR` version when you add functionality in a backwards-compatible manner, and
> 3. `PATCH` version when you make backwards-compatible bug fixes.[1]

The current version of the project must be manually maintained in the following two places:

- `/package.json` in property `"version"`
- `/bower.json` in property `"version"`

All other occurrences of the version (i.e. Docker images, packaged releases & the menu bar of the application itself) are resolved through the `"version"` property of `/package.json` automatically.

Pull requests

Using Git-Flow means that PRs have the highest chance of getting accepted and merged when you open them on the `develop` branch of your fork. That allows for some post-merge changes by the team without directly compromising the `master` branch, which is supposed to hold always be in a release-ready state.

It is usually not a big deal if you accidentally open a PR for the `master` branch. GitHub added the possibility to change the target branch for a PR afterwards some time ago.

Contribution guidelines

The minimum requirements for code contributions are:

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

JavaScript standard style guide

Since v2.7.0 the `npm test` script also verifies code compliance with the `standard` style before running the tests. If PRs deviate from this coding style, they will now immediately fail their build and will not be merged until compliant.

JS Standard Code Style

In case your PR is failing from style guide issues try running `standard --fix` over your code - this will fix all syntax or code style issues automatically without breaking your code. You might need to `npm i -g standard` first.

Testing

```
npm test            # check for code style violations and run all unit tests
npm run frisby      # run all API integration tests
npm run protractor # run all end-to-end tests
```

Pull Requests are verified to pass all of the following test stages during the continuous integration build. It is recommended that you run these tests on your local computer to verify they pass before submitting a PR. New features should be accompanied by an appropriate number of corresponding tests to verify they behave as intended.

Unit tests

There is a full suite containing isolated unit tests

- for all client-side code in `test/client`
- for the server-side routes and libraries in `test/server`

```
npm test
```

Integration tests

The integration tests in `test/api` verify if the backend for all normal use cases of the application works. All server-side vulnerabilities are also tested.

```
npm run frisby
```

These tests automatically start a server and run the tests against it. A working internet connection is recommended.

End-to-end tests

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

```
npm run protractor
```

The end-to-end tests require a locally installed Google Chrome browser and internet access to be able to pass.

Mutation tests

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

```
npm run stryker
```

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

Continuous integration & deployment

Travis-CI

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

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

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

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

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

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

AppVeyor

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

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

Testing packaged distrubutions

During releases the application will be packaged into `.zip` / `.tgz` archives for another easy setup method. When you contribute a change that impacts what the application needs to include, make sure you test this manually on your system.

```
grunt package
```

Then take the created archive from `/dist` and follow the steps described above in Packaged Distributions to make sure nothing is broken or missing.

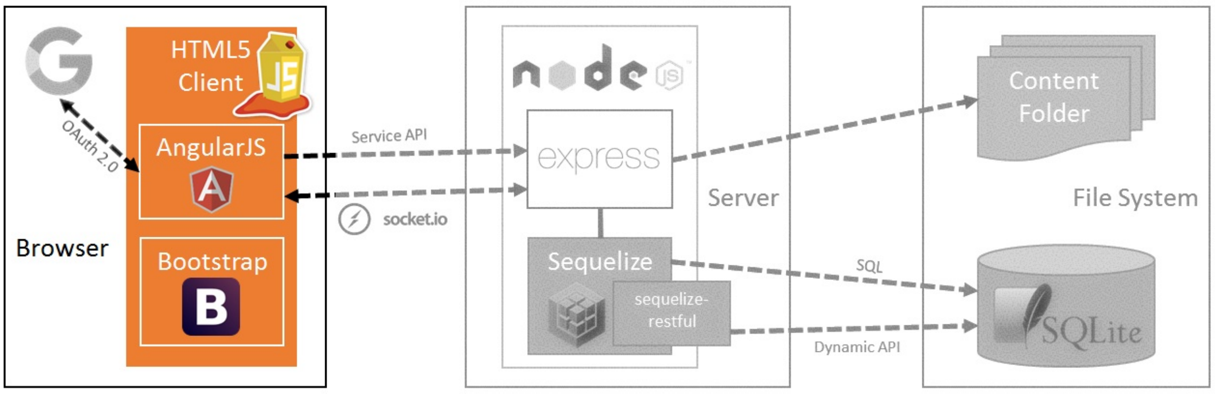
> 1. http://semver.org ↩

Codebase 101

Jumping head first into any foreign codebase can cause a little headache. This section is there to help you find your way through the code of OWASP Juice Shop. On its top level the Juice Shop codebase is mainly separated into a client and a server tier, the latter with an underlying lightweight database and file system as storage.

Client Tier

OWASP Juice Shop uses v1.5 of the popular AngularJS framework as the core of its client-side. Thanks to the Bootstrap CSS framework, the UI is responsive letting it adapt nicely to different screen sizes. Both frameworks work very well together thanks to the UI Bootstrap library that provides components for Bootstrap that are actually written in pure AngularJS. The various icons used throughout the frontend are from the vast Font Awesome 4 collection.



Services

TODO

Controllers

TODO

Views

TODO

Index page template

TODO

Client-side code minification

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

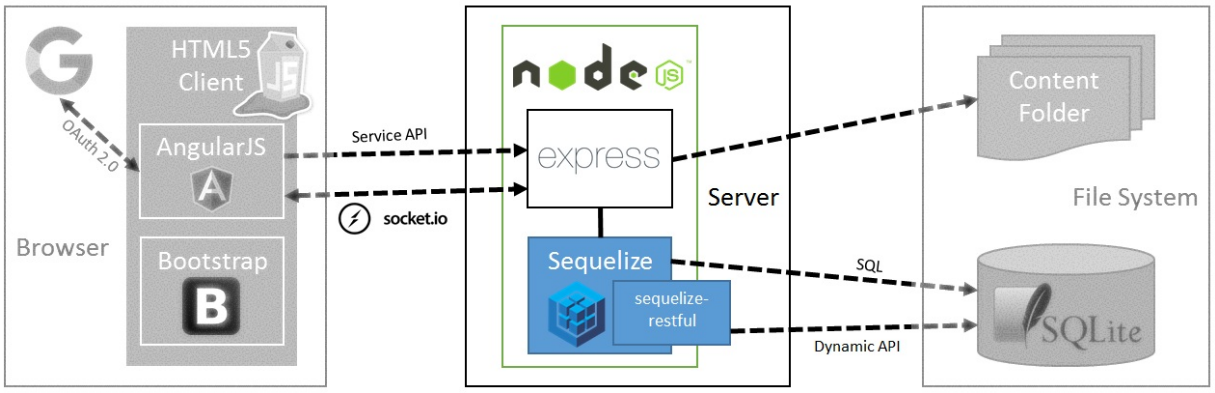
If you want to quickly test client-side code changes, it can be cumbersome to launch `npm install` over and over again. Instead you can simply trigger the minification to generate the `juice-shop.min.js` file with

```
grunt minify
```

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

Server Tier

TODO



Routes

TODO

Generated API endpoints

TODO

Hand-written routes

TODO

Custom libraries

TODO

Useful utilities

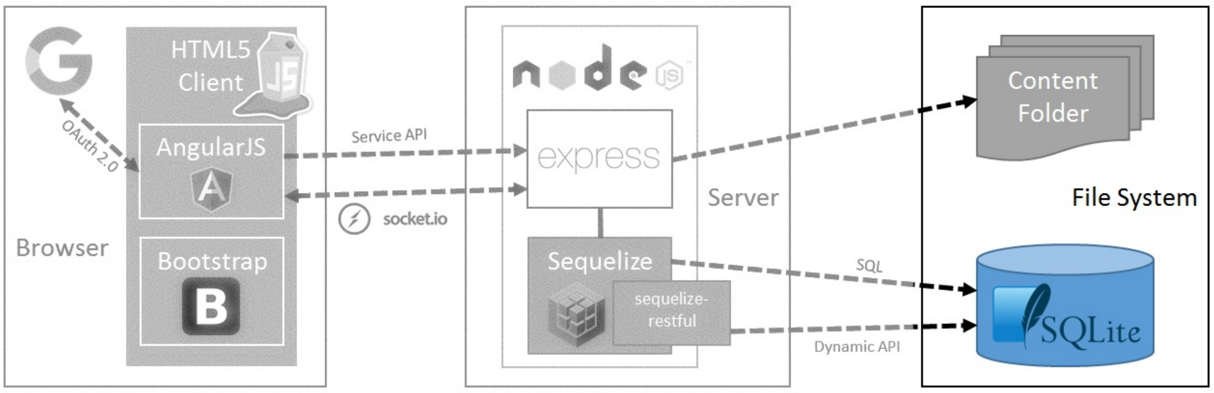
TODO

Insecurity features

TODO

Storage Tier

TODO



Database

TODO

Populating the DB

TODO

Non-relational database

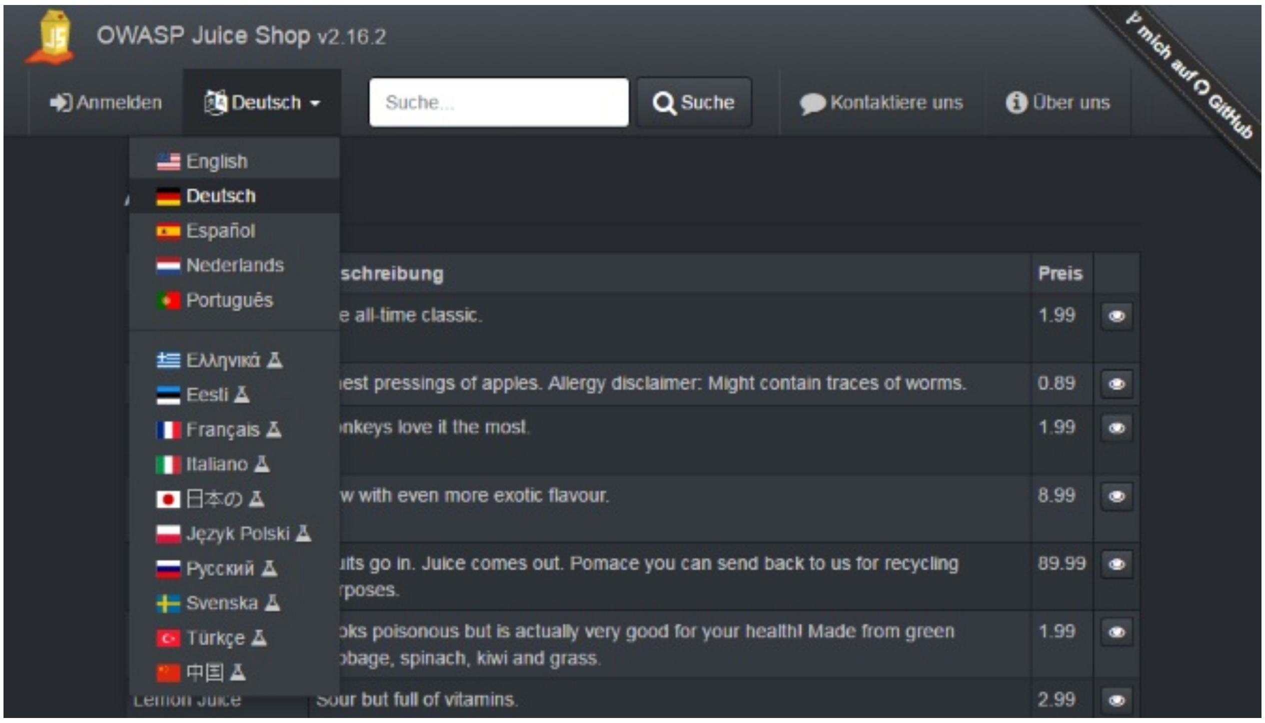
🔧 **TODO**

File system

🔧 **TODO**

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:



schreibung	Preis
e all-time classic.	1.99
est pressings of apples. Allergy disclaimer: Might contain traces of worms.	0.89
nkeys love it the most.	1.99
w with even more exotic flavour.	8.99
its go in. Juice comes out. Pomace you can send back to us for recycling rposes.	89.99
oks poisonous but is actually very good for your health! Made from green bbage, spinach, kiwi and grass.	1.99
Lemon Juice Sour but full of vitamins.	2.99

As long as the original author is taking part in the project's maintenance, there will always be 🇬🇧🇺🇸 **English** and a complete 🇩🇪 **German** translation available. Everything beyond that depends on volunteer translators!

Crowdin

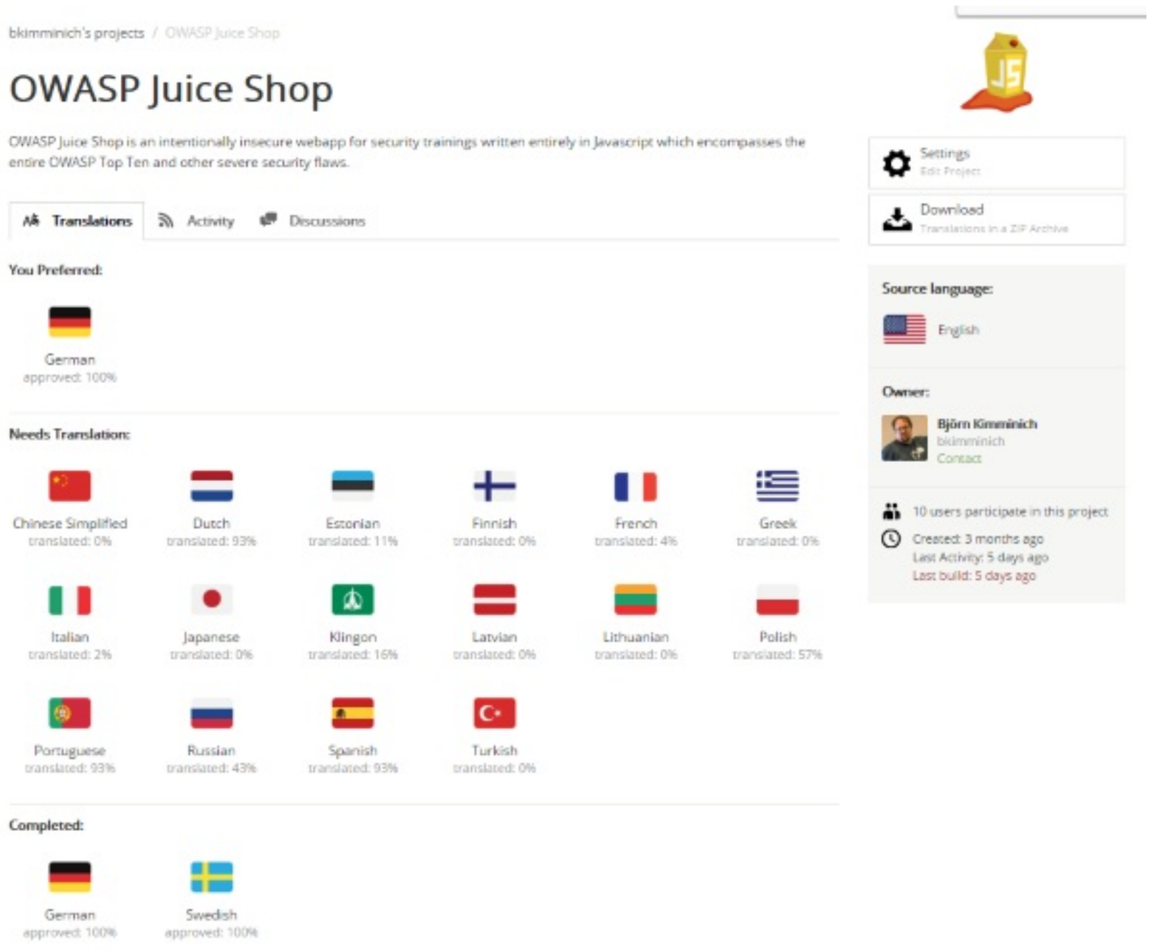
Juice Shop uses a Crowdin project to translate the project and perform reviews:

https://crowdin.com/project/owasp-juice-shop

Crowdin is a *Localization Management Platform* that allows to crowdsource translations of mobile apps, web, desktop software and related assets. It is free for open source projects.[1]

How to participate?

1. Create an account at Crowdin and log in.
2. Visit the project invitation page https://crowdin.com/project/owasp-juice-shop/invite
3. Pick a language you would like to help translate the project into



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.

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%

Source language: English

Owner: Björn Kimminich 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 `l10n_develop` Git branch to synchronize translations into the `app/i18n/??.json` language files where `??` is a language code (e.g. `en` or `de`).

Adding another language

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

Translating directly via GitHub PR

1. Fork the repository https://github.com/bkimminich/juice-shop
2. Translate the labels in the desired language- `.json` file in `/app/i18n`
3. Commit, push and open a Pull Request
4. Done!

If the language you would like to translate into is missing, just add a corresponding two-letter-ISO-code- `.json` file to the folder `/app/i18n` . It will be imported to Crowdin afterwards and added as a new language there as well.

The Crowdin process is the preferred way for the project to handle its translations as it comes with built-in review and approval options and is very easy to use. But of course it would be stupid of us to turn down a translation just because someone likes to edit JSON files manually more!

[1]. https://crowdin.com/ ↩

Donations

As a project of the OWASP Foundation the Juice Shop is and always will be

- open source
- free software

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

This does not mean that a project like it can thrive without any funding. Some examples on what the OWASP Juice Shop spent (or might spend) money on:

- Giveaways for conferences and meetups (e.g. stickers)
- 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)

How can I donate?

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

Provider	**Link**
PayPal (preferred)	Donate
Gratipay	https://gratipay.com/juice-shop
Flattr	https://flattr.com/thing/3856930/bkimminichjuice-shop-on-GitHub

BitCoin	1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm
Dash	Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW
Ethereum	0x0f933ab9fCAAA782D0279C300D73750e1311EAE6

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

> Please refer to the Donations section at the bottom of the project's `README.md` file on GitHub for possible additional options.

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