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
Log in with the administrator's user account.	⭐⭐
Log in with Bender's user account.	⭐⭐⭐
Log in with Jim's user account.	⭐⭐⭐
Order the Christmas special offer of 2014.	⭐⭐⭐
Find an old Recycle request and inform the shop about its unusual address. (Mention the entire delivery or pickup address in your comment)	⭐⭐⭐⭐
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.	⭐⭐⭐⭐
All your orders are belong to us!	⭐⭐⭐⭐⭐
Infect the server with malware by abusing arbitrary command execution.	⭐⭐⭐⭐⭐⭐

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.

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.

Order the Christmas special offer of 2014

> Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.
>
> When an attacker exploits SQL injection, sometimes the web application displays error messages from the database complaining that the SQL Query's syntax is incorrect. Blind SQL injection is nearly identical to normal SQL Injection, the only difference being the way the data is retrieved from the database. When the database does not output data to the web page, an attacker is forced to steal data by asking the database a series of true or false questions. This makes exploiting the SQL Injection vulnerability more difficult, but not impossible.[4]

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

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*.
- Neither of the above can be achieved through the application frontend and it might even require Blind SQL Injection.

Find an old Recycle request and inform the shop about its unusual address

🔧 **TODO**

Hints

🔧 **TODO**

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 gives away the expected attack vector for this challenge. Trying any others will not solve the challenge, even if they might yield the same result.
- In particular, flooding the application with requests will **not** solve this challenge. *That* would probably just *kill* your server instance.

Update multiple product reviews at the same time

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

Hints

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

Retrieve a list of all user credentials via SQL Injection

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

Hints

- Try to find an endpoint where you can influence data being retrieved from the server.
- Craft a `UNION SELECT` attack string to join data from another table into the original result.
- You might have to tackle some query syntax issues step-by-step, basically hopping from one error to the next
- As with Order the Christmas special offer of 2014 this cannot be achieved through the application frontend but involves some Blind SQL Injection instead.

All your orders are belong to us

TODO

Hints

TODO

Infect the server with malware by abusing arbitrary command execution

🔧 **TODO**

Hints

> 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 ↩

> 4. https://www.owasp.org/index.php/Blind_SQL_Injection ↩

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

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

You might have already solved this challenge along with Log in with the administrator's user account if you chose not to use SQL Injection. This challenge can only be solved if you use the original password of the administrator. If you changed the password previously, 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 the password of Bjoern's OWASP account via the Forgot Password mechanism

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

> Many website registrations use security questions for both password retrieval/reset and sign-in verification. Some also ask the same security questions when users call on the phone. Security questions are one method to verify the user and stop unauthorized access. But there are problems with security questions. Websites may use poor security questions that may have negative results:
>
> The user can't accurately remember the answer or the answer changed, The question doesn't work for the user, The question is not safe and could be discovered or guessed by others. It is essential that we use good questions. Good security questions meet five criteria. The answer to a good security question is:
>
> - **Safe**: cannot be guessed or researched
> - **Stable**: does not change over time
> - **Memorable**: can remember
> - **Simple**: is precise, easy, consistent
> - **Many**: has many possible answers
>
> It is difficult to find questions that meet all five criteria which means that some questions are good, some fair, and most are poor. **In reality, there are few if any GOOD security questions.** People share so much personal information on social media, blogs, and websites, that it is hard to find questions that meet the criteria above. In addition, many questions are not applicable to some people; for example, what is your oldest child's nickname – but you don't have a child.[1]

Hints

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

> Doxing (from dox, abbreviation of documents) or doxxing is the Internet-based practice of researching and broadcasting private or identifiable information (especially personally identifiable information) about an individual or organization.
>
> The methods employed to acquire this information include searching publicly available databases and social media websites (like Facebook), hacking, and social engineering. It is closely related to Internet vigilantism and hacktivism.
>
> Doxing may be carried out for various reasons, including to aid law enforcement, business analysis, risk analytics, extortion, coercion, inflicting harm, harassment, online shaming, and vigilante justice.[2]

Reset Jim's password via the Forgot Password mechanism

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

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

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

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

Hints

- The fact that the name of this challenge is "CSRF" is already a huge hint on how this challenge has to be approached.
- 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.[3]
>
> 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.[4]

Log in with Bjoern's Gmail account

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

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

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

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

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

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

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

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

Hints

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

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

> A software supply chain attack is when an attacker gains access to a legitimate software vendor and then compromises either the software or update repository. This is done with the intention of installing a backdoor, or other malicious code, into the legitimate software update provided by the vendor. As users update their software, unwittingly falling victim to the Trojanized update, they also install the embedded malicious code.[5]

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

Hints

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

> 1. http://goodsecurityquestions.com ↩

> 2. https://en.wikipedia.org/wiki/Doxing ↩

> 3. https://www.owasp.org/index.php/CSRF ↩

> 4. https://en.wikipedia.org/wiki/Rainbow_table ↩

5

> 5. https://www.rsa.com/en-us/blog/2017-02/are-software-supply-chain-attacks-the-new-norm ↩

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.	⭐⭐
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.
- More particular, they have been sloppy in a way that even the Angular Compiler was not able to clean up after them automatically.
- It is of course not sufficient to just visit 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.

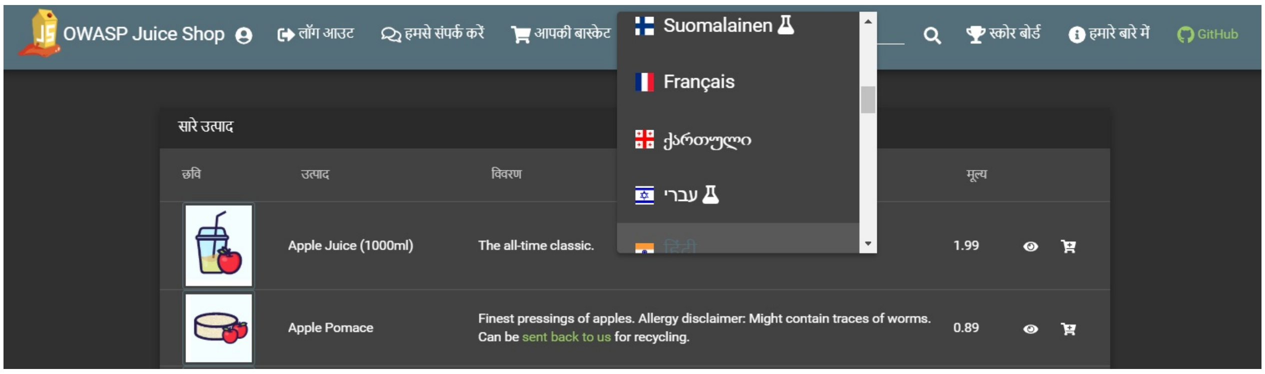
Hints

- 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.

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.



छवि	उत्पाद	विवरण	मूल्य
	Apple Juice (1000ml)	The all-time classic.	1.99
	Apple Pomace	Finest pressings of apples. Allergy disclaimer: Might contain traces of worms. Can be sent back to us for recycling.	0.89

Hints

- First you should find out how the languages are technically changed in the user interface.
- Guessing will most definitely not work in this challenge.
- You should rather choose between the following two ways to beat this challenge:
    - *Apply brute force* (and don't give up to quickly) to find it.
    - *Investigate online* what languages are actually available.

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

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

Hints

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

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

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

Roll your own Security

Challenges covered in this chapter

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

Behave like any "white hat" should

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

Hints

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

Submit 10 or more customer feedbacks within 10 seconds

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

> A completely automated public Turing test to tell computers and humans apart, or CAPTCHA, is a program that allows you to distinguish between humans and computers. First widely used by Alta Vista to prevent automated search submissions, CAPTCHAs are particularly effective in stopping any kind of automated abuse, including brute-force attacks. They work by presenting some test that is easy for humans to pass but difficult for computers to pass; therefore, they can conclude with some certainty whether there is a human on the other end.
>
> For a CAPTCHA to be effective, humans must be able to answer the test correctly as close to 100 percent of the time as possible. Computers must fail as close to 100 percent of the time as possible.[5]

Hints

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

Find the hidden easter egg

> An Easter egg is an intentional inside joke, hidden message, or feature in an interactive work such as a computer program, video game or DVD menu screen. The name is used to evoke the idea of a traditional Easter egg hunt.[1]

Hints

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

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

Access a developer's forgotten backup file

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

Hints

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

Access a misplaced SIEM signature file.

> Security information and event management (SIEM) technology supports threat detection and security incident response through the real-time collection and historical analysis of security events from a wide variety of event and contextual data sources. It also supports compliance reporting and incident investigation through analysis of historical data from these sources. The core capabilities of SIEM technology are a broad scope of event collection and the ability to correlate and analyze events across disparate sources.[2]

The misplaced signature file is actually a rule file for Sigma, a generic signature format for SIEM systems:

> Sigma is a generic and open signature format that allows you to describe relevant log events in a straight forward manner. The rule format is very flexible, easy to write and applicable to any type of log file. The main purpose of this project is to provide a structured form in which researchers or analysts can describe their once developed detection methods and make them shareable with others.
>
> Sigma is for log files what Snort is for network traffic and YARA is for files.[3]

Hints

- If you solved one of the other four file access challenges, you already know where the SIEM signature file is located
- Simply reuse the trick that already worked for the files above

Wherever you go, there you are

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

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

Hints

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

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

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

2. https://www.gartner.com/it-glossary/security-information-and-event-management-siem/ ↩

3. https://github.com/Neo23x0/sigma#what-is-sigma ↩

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

5. https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks#Sidebar:_Using_CAPTCHAS ↩

Sensitive Data Exposure

Challenges covered in this chapter

Challenge	Difficulty
Access a confidential document.	⭐
Log in with MC SafeSearch's original user credentials without applying SQL Injection or any other bypass.	⭐⭐
Inform the shop about an algorithm or library it should definitely not use the way it does.	⭐⭐
Log in with Amy's original user credentials. (This could take 93.83 billion trillion trillion centuries to brute force, but luckily she did not read the "One Important Final Note")	⭐⭐⭐
Gain access to any access log file of the server.	⭐⭐⭐⭐
Identify an unsafe product that was removed from the shop and inform the shop which ingredients are dangerous.	⭐⭐⭐⭐
Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to. (Creating a new account with the same password does not qualify as a solution.)	⭐⭐⭐⭐⭐
Perform an unwanted information disclosure by accessing data cross-domain.	⭐⭐⭐⭐⭐
Forge a coupon code that gives you a discount of at least 80%.	⭐⭐⭐⭐⭐⭐
Solve challenge #999. 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.

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

Hints

- Use the *Contact Us* form to submit a feedback mentioning the abused algorithm or library.
- There are five possible answers and you only need to identify one to solve the challenge.
- Cryptographic functions only used in the Apply some advanced cryptanalysis to find *the real* easter egg challenge *do not count* as they are only a developer's prank and not a serious security problem.

Log in with Amy's original user credentials

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

Hints

- As with so many other characters from Futurama this challenge is of course about logging in as Amy from that show. In the picture above you see her together with her alien husband Kif.
- The challenge description contains a few sentences which give away some information how Amy decided to strengthen her password.
- Obviously, Amy - being a little dimwitted - did not put nearly enough effort and creativity into the password selection process.

Gain access to any access log file of the server

> An access log is a list of all the requests for individual files that people have requested from a Web site. These files will include the HTML files and their imbedded graphic images and any other associated files that get transmitted. The access log (sometimes referred to as the "raw data") can be analyzed and summarized by another program.
>
> In general, an access log can be analyzed to tell you:
>
> The number of visitors (unique first-time requests) to a home page The origin of the visitors in terms of their associated server's domain name (for example, visitors from .edu, .com, and .gov sites and from the online services) How many requests for each page at the site, which can be presented with the pages with most requests listed first Usage patterns in terms of time of day, day of week, and seasonally Access log keepers and analyzers can be found as shareware on the Web or may come with a Web server.[2]

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

Hints

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

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

🔧 **TODO**

Hints

🔧 **TODO**

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

🔧 **TODO**

Hints

🔧 **TODO**

Perform an unwanted information disclosure by accessing data cross-domain

🔧 **TODO**

Hints

🔧 **TODO**

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 #999

The OWASP Juice Shop is *so broken* that even its convenience features (which have nothing to do with the e-commerce use cases) are designed to be vulnerable. One of these features is 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 #999 - 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 ↩

> 2. https://searchsecurity.techtarget.com/definition/access-log ↩

XML External Entities (XXE)

> An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.
>
> The XML 1.0 standard defines the structure of an XML document. The standard defines a concept called an entity, which is a storage unit of some type. There are a few different types of entities, external general/parameter parsed entity often shortened to external entity, that can access local or remote content via a declared system identifier. The system identifier is assumed to be a URI that can be dereferenced (accessed) by the XML processor when processing the entity. The XML processor then replaces occurrences of the named external entity with the contents dereferenced by the system identifier. If the system identifier contains tainted data and the XML processor dereferences this tainted data, the XML processor may disclose confidential information normally not accessible by the application. Similar attack vectors apply the usage of external DTDs, external stylesheets, external schemas, etc. which, when included, allow similar external resource inclusion style attacks.
>
> Attacks can include disclosing local files, which may contain sensitive data such as passwords or private user data, using file: schemes or relative paths in the system identifier. Since the attack occurs relative to the application processing the XML document, an attacker may use this trusted application to pivot to other internal systems, possibly disclosing other internal content via http(s) requests or launching a CSRF attack to any unprotected internal services. In some situations, an XML processor library that is vulnerable to client-side memory corruption issues may be exploited by dereferencing a malicious URI, possibly allowing arbitrary code execution under the application account. Other attacks can access local resources that may not stop returning data, possibly impacting application availability if too many threads or processes are not released.
>
> Note that the application does not need to explicitly return the response to the attacker for it to be vulnerable to information disclosures. An attacker can leverage DNS information to exfiltrate data through subdomain names to a DNS server that he/she controls.[1]

Challenges covered in this chapter

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

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

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

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

Hints

- You already found the leverage point for this challenge if you solved Use a deprecated B2B interface that was not properly shut down.
- This challenge sounds a lot harder than it actually is, which amplifies how bad the underlying vulnerability is.
- Doing some research on typical XEE attack patterns bascially gives away the solution for free.

Give the server something to chew on for quite a while

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

Hints

- The leverage point for this is obviously the same as for the XXE Tier 1 challenge above.
- You can only solve this challenge by keeping the server busy for >2sec with your attack.
- The effectiveness of attack payloads for this challenge might depend on the operating system the Juice Shop is running on.

1

> 1. https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing ↩

Improper Input Validation

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

Challenges covered in this chapter

Challenge	Difficulty
Give a devastating zero-star feedback to the store.	⭐
Get registered as admin user.	⭐⭐⭐
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

Get registered as admin user

The Juice Shop does not bother to separate administrative functionality into a deployment unit of its own. Instead, the cheapest solution was chosen by simply leaving then admin features in the web shop itself and (allegedly) demanding a higher level of access to use them.

Hints

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

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 online 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.	⭐⭐
Put an additional product into another user's shopping basket.	⭐⭐⭐
Post some feedback in another users name.	⭐⭐⭐
Post a product review as another user or edit any user's existing review.	⭐⭐⭐
Change the href of the link within the O-Saft product description into http://kimminich.de.	⭐⭐⭐
Request a hidden resource on server through server.	⭐⭐⭐⭐⭐⭐

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 probably just slightly harder to find and gain access to than the score board link
- There is some access control in place, but there are at least three ways to bypass it.

View another user's shopping basket

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

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!

Put an additional product into another user's shopping basket

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

Hints

- Check the HTTP traffic while placing products into your own shopping basket to find a leverage point.
- Adding more instances of the same product to someone elses basket does not qualify as a solution. The same goes for stealing from someone elses basket.
- This challenge requires a bit more sophisticated tampering than others of the same ilk.

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.

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

TODO

Hints

TODO

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

Request a hidden resource on server through server

🔧 **TODO**

Hints

🔧 **TODO**

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

Security Misconfiguration

Challenges covered in this chapter

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

Provoke an error that is not very gracefully handled

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

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

Hints

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

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

Access a salesman's forgotten backup file

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

Hints

- Analyze and tamper with links in the application that deliver a file directly.
- The file is not directly accessible because a security mechanism prevents access to it.
- You need to trick the security mechanism into thinking that the file has a valid file type.
- For this challenge there are *two approaches* to succeed with one being based on Security Misconfiguration and the other on a Roll Your Own Security-problem.

Reset Morty's password via the Forgot Password mechanism

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

> A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response. For the sake of efficiency, an attacker may use a dictionary attack (with or without mutations) or a traditional brute-force attack (with given classes of characters e.g.: alphanumerical, special, case (in)sensitive). Considering a given method, number of tries, efficiency of the system which conducts the attack, and estimated efficiency of the system which is attacked the attacker is able to calculate approximately how long it will take to submit all chosen predetermined values.[2]

Hints

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

Log in with the support team's original user credentials

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

Hints

- The support team is located in some low-cost country and the team structure fluctuates a lot due to people leaving for jobs with even just slightly better wages.
- To prevent abuse the password for the support team account is very strong.
- To allow easy access during an incident, the support team utilizes a 3rd party tool which every support engineer can access to get the current account password from.
- While it is also possible to use SQL Injection to log in as the support team, this will not solve the challenge.

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

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

Cross Site Scripting (XSS)

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

Challenges covered in this chapter

Challenge	Difficulty
Perform a *reflected* XSS attack with `<iframe src="javascript:alert(` xss `)">`.	⭐
Perform a *DOM* XSS attack with `<iframe src="javascript:alert(` xss `)">`.	⭐
Perform an XSS attack with `<script>alert(` xss `)</script>` on a legacy page within the application.	⭐⭐
Perform a *persisted* XSS attack with `<iframe src="javascript:alert(` xss `)">` bypassing a client-side security mechanism.	⭐⭐⭐
Perform a *persisted* XSS attack with `<iframe src="javascript:alert(` xss `)">` without using the frontend application at all.	⭐⭐⭐
Perform a *persisted* XSS attack with `<iframe src="javascript:alert(` xss `)">` bypassing a server-side security mechanism.	⭐⭐⭐⭐
Perform a *persisted* XSS attack with `<iframe src="javascript:alert(` xss `)">` through an HTTP header.	⭐⭐⭐⭐

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 when its form is submitted.
- Try probing for XSS vulnerabilities by submitting text wrapped in an HTML tag which is easy to spot on screen, e.g. `<h1>` or `<strike>` .

Perform a DOM XSS attack

> DOM-based Cross-Site Scripting is the de-facto name for XSS bugs which are the result of active browser-side content on a page, typically JavaScript, obtaining user input and then doing something unsafe with it which leads to execution of injected code.
>
> The DOM, or Document Object Model, is the structural format used to represent documents in a browser. The DOM enables dynamic scripts such as JavaScript to reference components of the document such as a form field or a session cookie. The DOM is also used by the browser for security - for example to limit scripts on different domains from obtaining session cookies for other domains. A DOM-based XSS vulnerability may occur when active content, such as a JavaScript function, is modified by a specially crafted request such that a DOM element that can be controlled by an attacker.[3]

Hints

- This challenge is almost indistinguishable from Perform a reflected XSS attack if you do not look "under the hood" to find out what the application actually does with the user input

Perform an XSS attack on a legacy page within the application

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

Hints

- Find a screen in the application that looks subtly odd and dated compared with all other screens
- What is *even better* than homegrown validation based on a RegEx? Homegrown sanitization based on a RegEx!

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

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

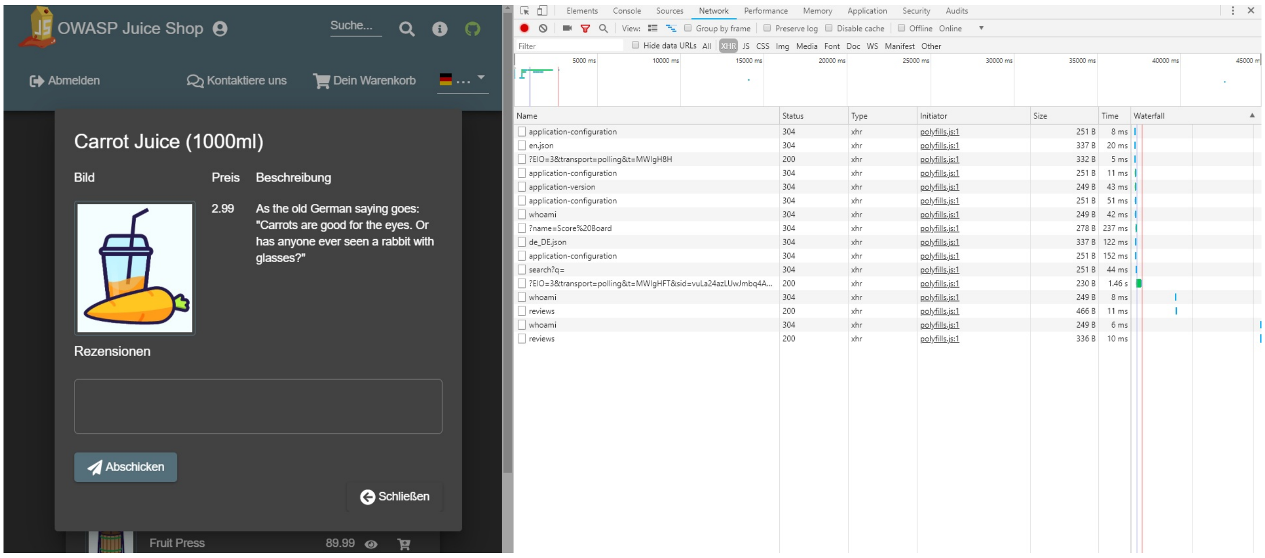
> Be aware that any JavaScript input validation performed on the client can be bypassed by an attacker that disables JavaScript or uses a Web Proxy. Ensure that any input validation performed on the client is also performed on the server.[4]

Hints

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

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

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



2.99 As the old German saying goes: "Carrots are good for the eyes. Or has anyone ever seen a rabbit with glasses?"

Name	Status	Type	Initiator	Size	Time
application-configuration	304	xhr	polyfills.js:1	251 B	8 ms
en.json	304	xhr	polyfills.js:1	337 B	20 ms
?EIO=3&transport=polling&t=MWIgH8H	200	xhr	polyfills.js:1	332 B	5 ms
application-configuration	304	xhr	polyfills.js:1	251 B	11 ms
application-version	304	xhr	polyfills.js:1	249 B	43 ms
application-configuration	304	xhr	polyfills.js:1	251 B	51 ms
whoami	304	xhr	polyfills.js:1	249 B	42 ms
?name=Score%20Board	304	xhr	polyfills.js:1	278 B	237 ms
de_DE.json	304	xhr	polyfills.js:1	337 B	122 ms
application-configuration	304	xhr	polyfills.js:1	251 B	152 ms
search?q=	304	xhr	polyfills.js:1	251 B	44 ms
?EIO=3&transport=polling&t=MWIgHFT&sid=vuLa24azLUwJmbq4A...	200	xhr	polyfills.js:1	230 B	1.46 s
whoami	304	xhr	polyfills.js:1	249 B	8 ms
reviews	200	xhr	polyfills.js:1	466 B	11 ms
whoami	304	xhr	polyfills.js:1	249 B	6 ms
reviews	200	xhr	polyfills.js:1	336 B	10 ms

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

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 `<iframe src="javascript:alert(` xss `)">` will *not be rejected* by any validator but *stripped from the comment* before persisting it

- Look for possible dependencies related to input processing in the `package.json.bak` you harvested earlier
- If an `xss` alert shows up but the challenge does not appear as solved on the *Score Board*, you might not have managed to put the *exact* attack string `<iframe src="javascript:alert(`xss`)">` into the database?

Perform a persisted XSS attack through an HTTP header

TODO

Hints

TODO

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

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

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

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

Insecure Deserialization

> Serialization is the process of turning some object into a data format that can be restored later. People often serialize objects in order to save them to storage, or to send as part of communications. Deserialization is the reverse of that process -- taking data structured from some format, and rebuilding it into an object. Today, the most popular data format for serializing data is JSON. Before that, it was XML.
>
> However, many programming languages offer a native capability for serializing objects. These native formats usually offer more features than JSON or XML, including customizability of the serialization process. Unfortunately, the features of these native deserialization mechanisms can be repurposed for malicious effect when operating on untrusted data. Attacks against deserializers have been found to allow denial-of-service, access control, and remote code execution attacks.[1]

Challenges covered in this chapter

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

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

> Code Injection is the general term for attack types which consist of injecting code that is then interpreted/executed by the application. This type of attack exploits poor handling of untrusted data. These types of attacks are usually made possible due to a lack of proper input/output data validation, for example:
>
> - allowed characters (standard regular expressions classes or custom)
> - data format
> - amount of expected data
>
> Code Injection differs from Command Injection in that an attacker is only limited by the functionality of the injected language itself. If an attacker is able to inject PHP code into an application and have it executed, he is only limited by what PHP is capable of. Command injection consists of leveraging existing code to execute commands, usually within the context of a shell.[2]
>
> The ability to trigger arbitrary code execution from one machine on another (especially via a wide-area network such as the Internet) is often referred to as remote code execution.[3]

Hints

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

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

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

Hints

- This challenge uses the same leverage point as Perform a Remote Code Execution that would keep a less hardened application busy forever.

- The application has a protection against too many iterations (i.e. *infinite loops*) which your attack must not trigger in order to solve this challenge.

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

> 2. https://www.owasp.org/index.php/Code_Injection ↩

> 3. https://en.wikipedia.org/wiki/Arbitrary_code_execution ↩

> 4. https://en.wikipedia.org/wiki/Infinite_loop ↩

Vulnerable Components

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

Challenges covered in this chapter

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

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

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

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

Hints

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

Inform the shop about a vulnerable library it is using

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

Hints

- Use the *Contact Us* form to submit a feedback mentioning the vulnerable library including its exact version.
- Look for possible dependencies related to security in the `package.json.bak` you probably harvested earlier during the Access a developer's forgotten backup file challenge.
- Do some research on the internet for known security issues in the most suspicious application dependencies.

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.
- 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 sneaky 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.

Overwrite the Legal Information file

> Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.
>
> The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.
>
> There are really two classes of problems here. The first is with the file metadata, like the path and file name. These are generally provided by the transport, such as HTTP multi-part encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. You must validate the metadata extremely carefully before using it.
>
> The other class of problem is with the file size or content. The range of problems here depends entirely on what the file is used for. See the examples below for some ideas about how files might be misused. To protect against this type of attack, you should analyse everything your application does with files and think carefully about what processing and interpreters are involved.[3]

Hints

- Find all places in the application where file uploads are possible.
- For at least one of these, the Juice Shop is depending on a library that suffers from an arbitrary file overwrite vulnerability.

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 ↩

> 3. https://www.owasp.org/index.php/Unrestricted_File_Upload ↩

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
Apply some advanced cryptanalysis to find *the real* easter egg.	⭐⭐⭐⭐
Rat out a notorious character hiding in plain sight in the shop.	⭐⭐⭐⭐
Learn about the Token Sale before its official announcement.	⭐⭐⭐⭐⭐

Apply some advanced cryptanalysis to find the real easter egg

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

Hints

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

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

> Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. The word steganography combines the Greek words steganos (στεγανός), meaning "covered, concealed, or protected", and graphein (γράφειν) meaning "writing".
>
> The first recorded use of the term was in 1499 by Johannes Trithemius in his Steganographia, a treatise on cryptography and steganography, disguised as a book on magic. Generally, the hidden messages appear to be (or to be part of) something else: images, articles, shopping lists, or some other cover text. For example, the hidden message may be in invisible ink between the visible lines of a private letter. Some implementations of steganography that lack a shared secret are forms of security through obscurity, and key-dependent steganographic schemes adhere to Kerckhoffs's principle.
>
> The advantage of steganography over cryptography alone is that the intended secret message does not attract attention to itself as an object of scrutiny. Plainly visible encrypted messages, no matter how unbreakable they are, arouse interest and may in themselves be incriminating in countries in which encryption is illegal.
>
> Whereas cryptography is the practice of protecting the contents of a message alone, steganography is concerned with concealing the fact that a secret message is being sent as well as concealing the contents of the message.
>
> Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a transport layer, such as a document file, image file, program or protocol. Media files are ideal for steganographic transmission because of their large size. For example, a sender might start with an innocuous image file and adjust the color of every hundredth pixel to correspond to a letter in the alphabet. The change is so subtle that someone who is not specifically looking for it is unlikely to notice the change.[3]

Hints

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

Learn about the Token Sale before its official announcement

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

> An initial coin offering (ICO) is a controversial means of crowdfunding centered around cryptocurrency, which can be a source of capital for startup companies. In an ICO, a quantity of the crowdfunded cryptocurrency is preallocated to investors in the form of "tokens", in exchange for legal tender or other cryptocurrencies such as bitcoin or ethereum. These tokens supposedly become functional units of currency if or when the ICO's funding goal is met and the project launches.
>
> ICOs provide a means by which startups avoid costs of regulatory compliance and intermediaries, such as venture capitalists, bank and stock exchanges, while increasing risk for investors. ICOs may fall outside existing regulations or may need to be regulated depending on the nature of the project, or are banned altogether in some jurisdictions, such as China and South Korea.
>
> [...] The term may be analogous with "token sale" or crowdsale, which refers to a method of selling participation in an economy, giving investors access to the features of a particular project starting at a later date. ICOs may sell a right of ownership or royalties to a project, in contrast to an initial public offering which sells a share in the ownership of the company itself.[2]

Hints

- Guessing or brute forcing the URL of the token sale page is very unlikely to succeed.
- You should closely investigate the place where all paths within the application are defined.
- Beating the employed obfuscation mechanism manually will take some time. Maybe there is an easier way to undo it?

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

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

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

Race Condition

> A race condition or race hazard is the behavior of an electronics, software, or other system where the output is dependent on the sequence or timing of other uncontrollable events. It becomes a bug when events do not happen in the order the programmer intended.[1]

> Many software race conditions have associated computer security implications. A race condition allows an attacker with access to a shared resource to cause other actors that utilize that resource to malfunction, resulting in effects including denial of service and privilege escalation.
>
> A specific kind of race condition involves checking for a predicate (e.g. for authentication), then acting on the predicate, while the state can change between the time of check and the time of use. When this kind of bug exists in security-sensitive code, a security vulnerability called a time-of-check-to-time-of-use (TOCTTOU) bug is created.[2]

Challenges covered in this chapter

Challenge	Difficulty
Like any review at least three times as the same user.	⭐⭐⭐⭐⭐⭐

Like any review at least three times as the same user

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

Hints

- Every user is (almost) immediately associated with the review they "liked" to prevent abuse of that functionality
- Did you really think clicking the "like" button three times in a row *really fast* would be enough to solve a ⭐⭐⭐⭐⭐⭐ challenge?

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

> 2. https://en.wikipedia.org/wiki/Race_condition#Computer_security ↩