

Capstone: Web Security and OWASP

Daniel Pepin

CSCI 401

Prof. Holst

INTRODUCTION

The topic of web security is one that is constantly changing. Although web technologies evolve and are continuously being improved they are also broken down at the same pace. Because of this, cybersecurity professionals and institutions publish frameworks and standards for developers in order to raise awareness and mitigate the many security issues they face while creating projects on the web. One such foundation is OWASP (The Open Web Application Security Project). OWASP is a nonprofit organization that seeks to improve the security of software through education. Some of their projects include *OWASP Top Ten*: a document that raises awareness of the most critical security risks for web applications, *OWASP Cheat Sheet*: a series meant to provide concise yet valuable information on specific security topics, and *OWASP Juice Shop*: a purposefully insecure web application used to train developers on common vulnerabilities found in the wild. This paper will use *OWASP Juice Shop* to showcase some of the common vulnerabilities listed in the *OWASP Top Ten* list and later explain how to mitigate these attacks.

OWASP TOP TEN

The OWASP Top Ten is a living document that lists the most critical security risks to web applications. The list is meant to serve as the first step for developers towards a more secure application. The list is compiled and published every three to four years [6]. As of the writing of this paper, the list is currently numbered as follows: 1. Injection, 2. Broken authentication, 3. Sensitive data exposure, 4. XML external entities (XXE), 5. Broken access control, 6. Security misconfigurations, 7. Cross-site scripting (XSS), 8. Insecure deserialization, 9. Using components with known vulnerabilities, 10. Insufficient logging and monitoring. Not all vulnerabilities are built equally. Some carry different levels of severity, can be harder to carry

out, or require victim interaction. Websites can collect incredibly valuable information about their users ranging from names, emails, and passwords. Some of this information, like passwords, are sensitive and often reused. Google published a study that finds two in three users reuse the same password. This information is sometimes leaked out by hackers whose ransom demands were not met. Using OWASP top 10 a white paper was published detailing the process of reconnaissance, scanning, and attempts at exploiting a website to understand its level of security. Using various tools like nmap, The Harvester, OWASP Zap, and many more, they attempted all 10 of what is detailed on OWASP top 10 to gauge the level of safety on a website and sub-domains. “Based on the results of security test research submitted on OWASP 10, The STMIK Sumedang website has a security level of 80%, a subdomain of Informatics Study Program has a security level of 60%” (“Analysis of Web Security Using Open Web Application Security Project 10, 2021). The initial website was given an 80 percent because of what it was able to cover on the top 10 list. While it was protecting itself from many of the issues detailed, the website was found to have a lot of information about its backend leaked and even found open ports that could be used as attack vectors. It’s important to be able to not disclose information and error details and also at least implement network logging and login features when it comes to open ports.

INFORMATION STORAGE

When signing up for a website, your information must be stored for later retrieval. This information is stored in a database on another computer. This computer is called the server. When signing into a website, a lookup for your information must be done. The website needs to find your email and make sure your password matches. In order to do this, their server must

communicate with the database using a language it understands. One such language is SQL (Structured Query Language). A server is able to send commands known as queries to request, update, create, or destroy information. In a SQL database information is stored in tables. A table can be thought of as a bin full of documents, These documents are referred to as rows.

```
sqlite> CREATE TABLE users(username varchar(25), password varchar(256));
sqlite> INSERT INTO users VALUES("Admin", "abc123");
sqlite> SELECT * FROM users;
Admin|abc123
sqlite> █
```

ATTACK VECTORS

With how many moving parts there are for web applications and search engines, naturally there will be many attack vectors to protect. Attack vectors are methods in which adversaries can use to breach, steal information, and even damage the software/ hardware they are targeting.

According to OWASP Top Ten, a standard awareness document for developers, they provide 10 security risks that they feel represents a broad consensus as to what is most crucial to the security of web applications. In this section I will go over a couple that we will use to attack the juice shop; Injection, “Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query.” (*OWASP Top Ten* 2021). With this an attacker can use a domain-specific language like NoSQL or other versions of SQL to exploit input fields on a page causing the interpreter to execute such commands allowing the attacker to gain access to tables in a database with whatever information that table contains or even re write some fields in the tables held server side. Seventh on this list, Cross-Site Scripting (XSS) can occur, “ whenever an application includes untrusted data in a new web page without

proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript” (*OWASP Top Ten* 2021). This can allow an attacker to cause websites to execute scripts on other users, or redirect users to malicious sites. Third on this list is Sensitive Data Exposure which is the issue of “ web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes.” (*OWASP Top Ten* 2021). This is an important concept to try and cover as this is not just an attack vector but could be a result of a different attack vector that can cause a web application to hemorrhage data and even more ways of attacking it. Second on this list is Broken Authentication, which allows “attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently.” (*OWASP Top Ten* 2021). Methods to exploit Broken Authentication, can range as complex as cookie manipulation or as simple as brute forcing someone's password allowing you to log in as someone else. Understanding these attack vectors is imperative to mitigation of damage to be done on web applications.

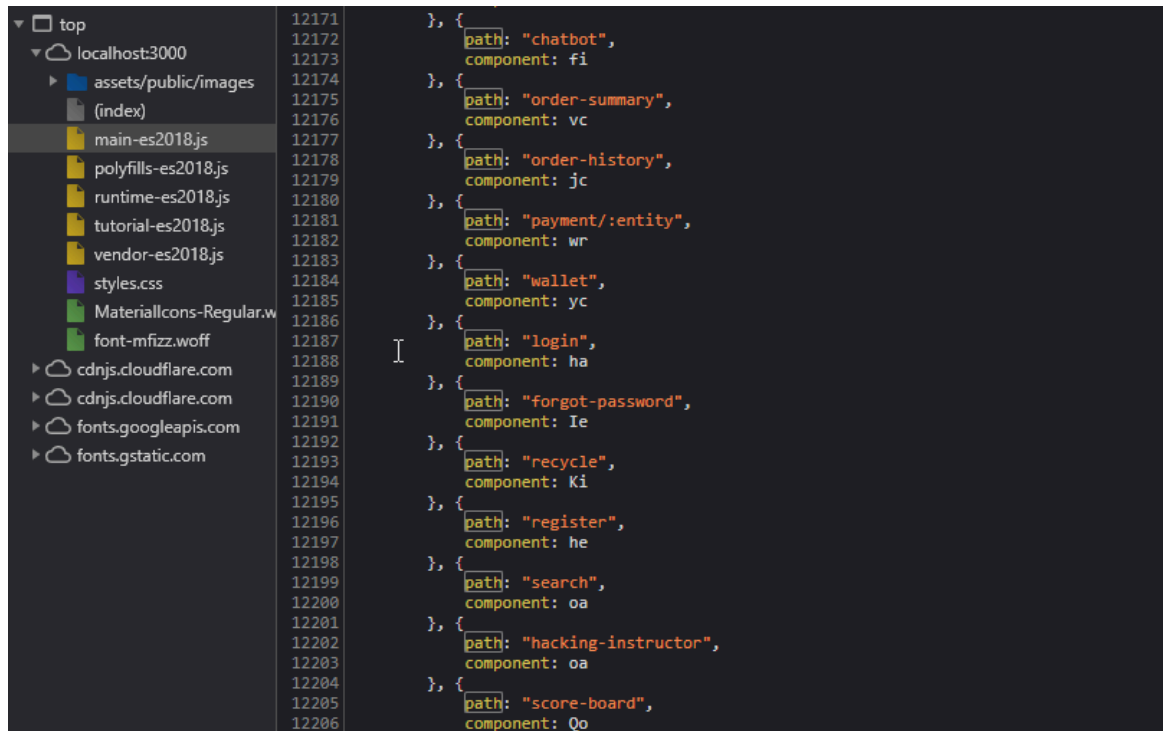
Anatomy of Attacks

_____ In order to get to performing attacks such as SQL injections and Cross Site Scripting attacks, a more in depth inspection and understanding it in a technical matter is required. A whitepaper on the anatomy of an XSS attack provides great insight on how XSS attacks work, beginning with breaking down the different types of XSS attacks. There are three types of XSS attacks: DOM based XSS attacks, Stored XSS, and Reflected XSS attacks. “Domed based XSS is also known as type 0 XSS. It is a client side attack, in this attack user provided data is written

to the document object model (DOM)” (A comprehensive inspection of cross site scripting attack, 2021). “Stored XSS is also known as persistent XSS or type 1 XSS. In this attack, the attacker embedded a malicious script which gets permanently stored on the server” (A comprehensive inspection of cross site scripting attack, 2021). “Reflected XSS is also known as non persistent or type 2 XSS. In this attack the attacker first builds the malicious link. After creating the malicious link it sends the URL to the user via email and persuades them to click on it” (A comprehensive inspection of cross site scripting attack, 2021). In terms of SQL attacks it’s important to understand the basic usage of such queries. “In the process of SQL injection attack, the attacker inserts malicious code fragments into the request parameters, which causes the server to execute illegal queries, resulting in data leakage and database damage” (A SQL Injection Detection Method Based on Adaptive Deep Forest, 2021). To form SQL injections we need to target user input sections where we may be able to cause a return in information like which SQL language is it so we can form proper queries.

OWASP Juice Shop

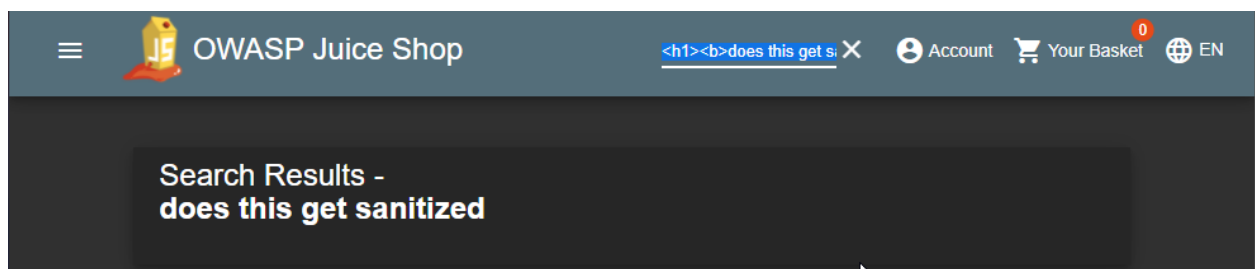
For OWASP Juice Shop, there are a couple ways to use the site to practice attacks. The way I will be presenting it is from my local machine from pulling the source from git and using Node Js in order to compile and run the site. Initially upon loading the site on our local host, we’re presented with the first challenge to find the scoreboard. There are a couple ways to find this endpoint; some may try to brute force it by adding a scoreboard or other variants of that string into the end of the url. Instead of doing that a really important thing to do is get used to the developer views for chrome or any other browser.



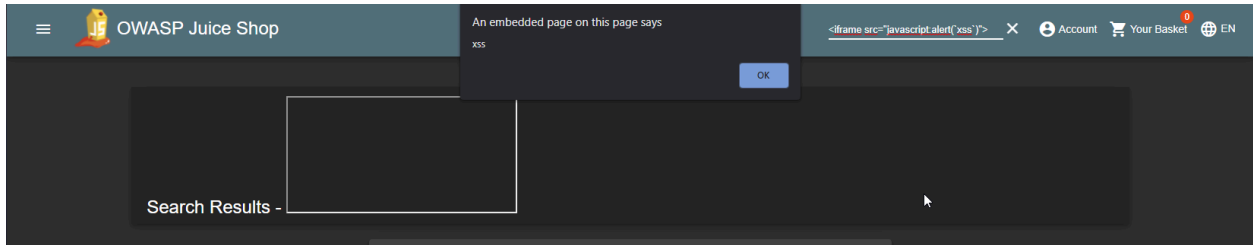
```
12171 }, {
12172   path: "chatbot",
12173   component: fi
12174 }, {
12175   path: "order-summary",
12176   component: vc
12177 }, {
12178   path: "order-history",
12179   component: jc
12180 }, {
12181   path: "payment/:entity",
12182   component: wr
12183 }, {
12184   path: "wallet",
12185   component: yc
12186 }, {
12187   path: "login",
12188   component: ha
12189 }, {
12190   path: "forgot-password",
12191   component: le
12192 }, {
12193   path: "recycle",
12194   component: ki
12195 }, {
12196   path: "register",
12197   component: he
12198 }, {
12199   path: "search",
12200   component: oa
12201 }, {
12202   path: "hacking-instructor",
12203   component: oa
12204 }, {
12205   path: "score-board",
12206   component: Qo
```

Using the developer tools allows us to view the sources of the site, here we can see in main.js, all of the paths that are available within the site. Not only did we find the path score-board at the bottom we now have many other endpoints to our disposal.

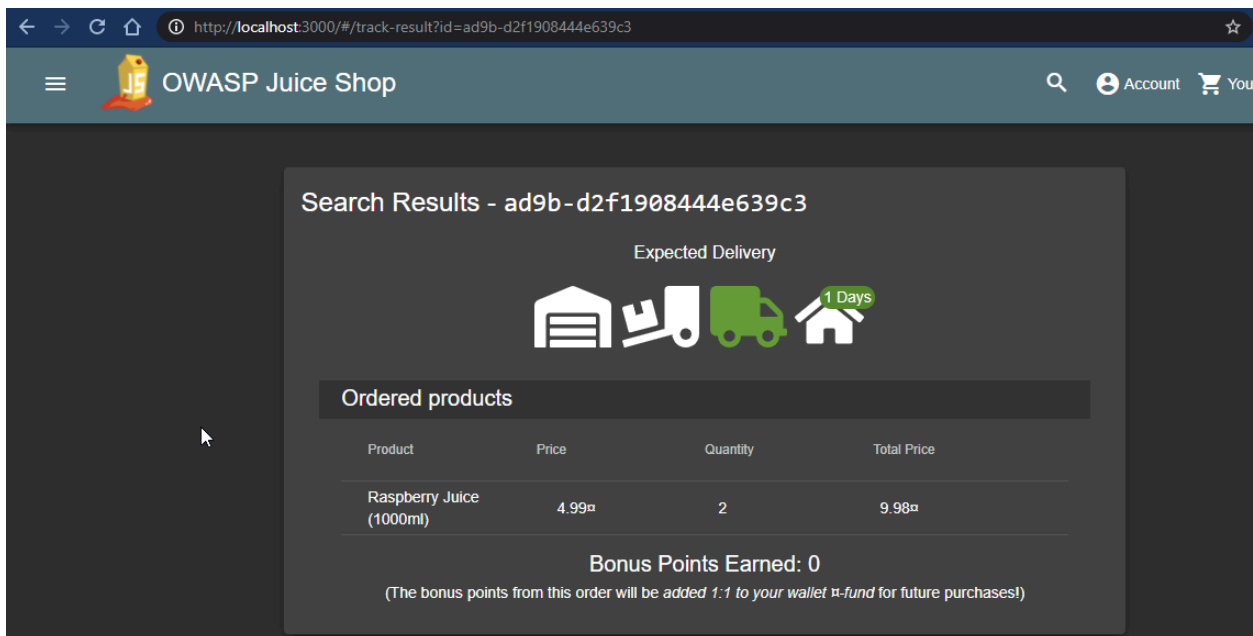
For our first attempt at an XSS attack I wanted to look for possible places of input where I could inject some basic html to see if it gets sanitized or not.



While this is not our XSS attack just yet, this gives us great information. By inputting “<h1>does this get sanitized</h1>” into the search bar we get search results back with what we entered on top, and as we can see the string in between the tags are bolded meaning that our “” tags are not getting sanitized same goes for the h1 tags.



With the previously found knowledge, we can now try and inject something that would execute javascript. "<iframe src='\"javascript:alert('\"xss')\"'>\"", with this iframe tag which embeds a new document into the currently existing html document we can use javascript:alert function as a source causing it to be executed on the page. Now with this basic XSS attack we see that we can inject code into different places on the site, but with this one in particular won't affect other users on the site as this XSS is in the search bar and is not stored in a way where another user can access this. Next we'll take a look at the ordering system that is used by the store and the system used to track orders.



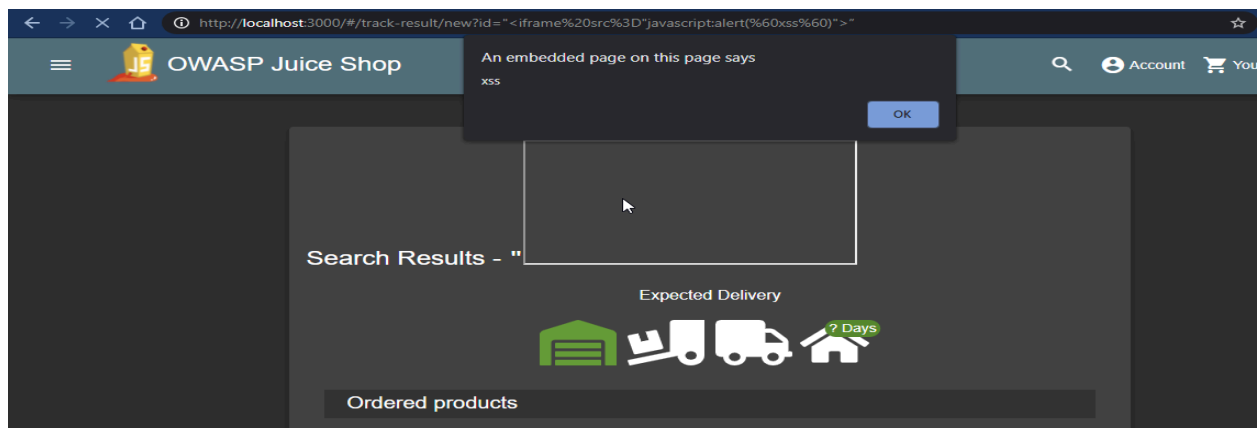
Here we have the expected delivery page with what looks to be a tracking id. Considering the current url queries and id with "?id", we may be able to do something to that url in order to

create a reflected XSS attack. If we take the current url

“`http://localhost:3000/#/track-result?id=ad9b-d2f1908444e639c3`” and take our previous script

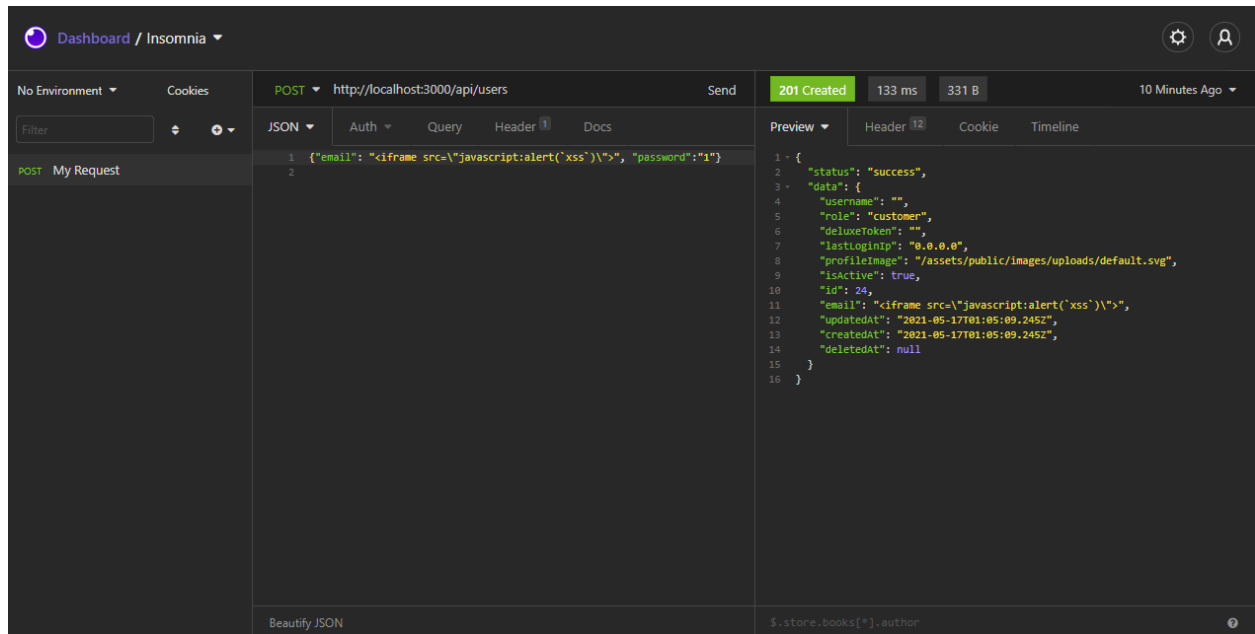
we used to create an XSS we can come up with something like this,

“`http://localhost:3000/#/track-result?id= <iframe src="javascript:alert(`xss`)">`”. Replacing the original url like this will cause that iframe to be rendered where the id gets rendered onto the page.

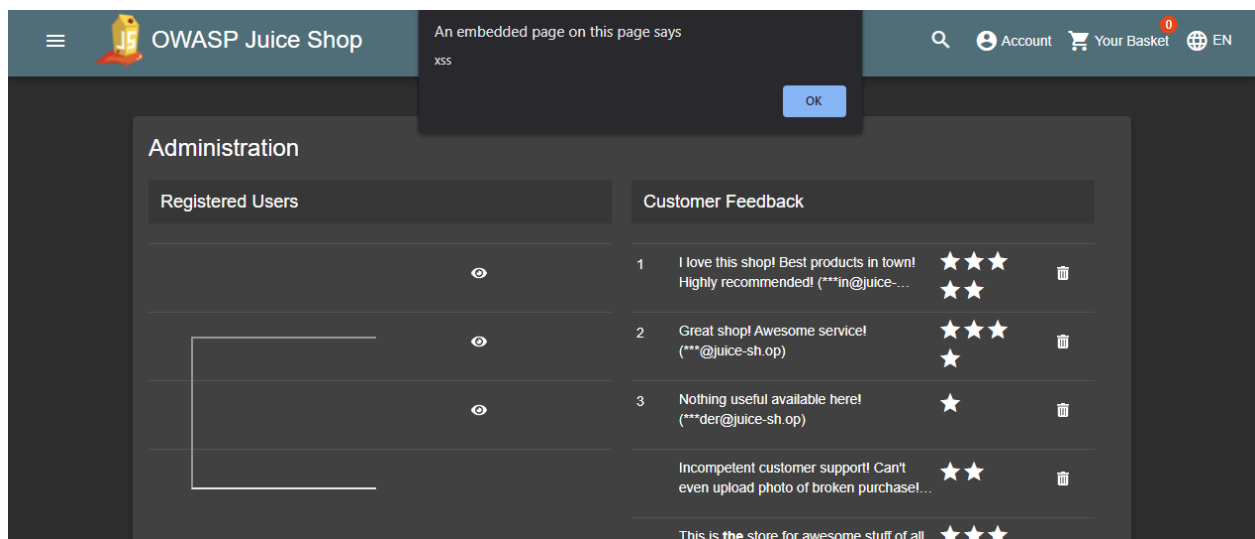


With this simple yet crafty manipulation of the javascript on the site could even cause a user's cookies to be stolen or download a malicious package. Instead of alerting xss onto the screen we could call upon `document.cookie` to show the users cookies like this,

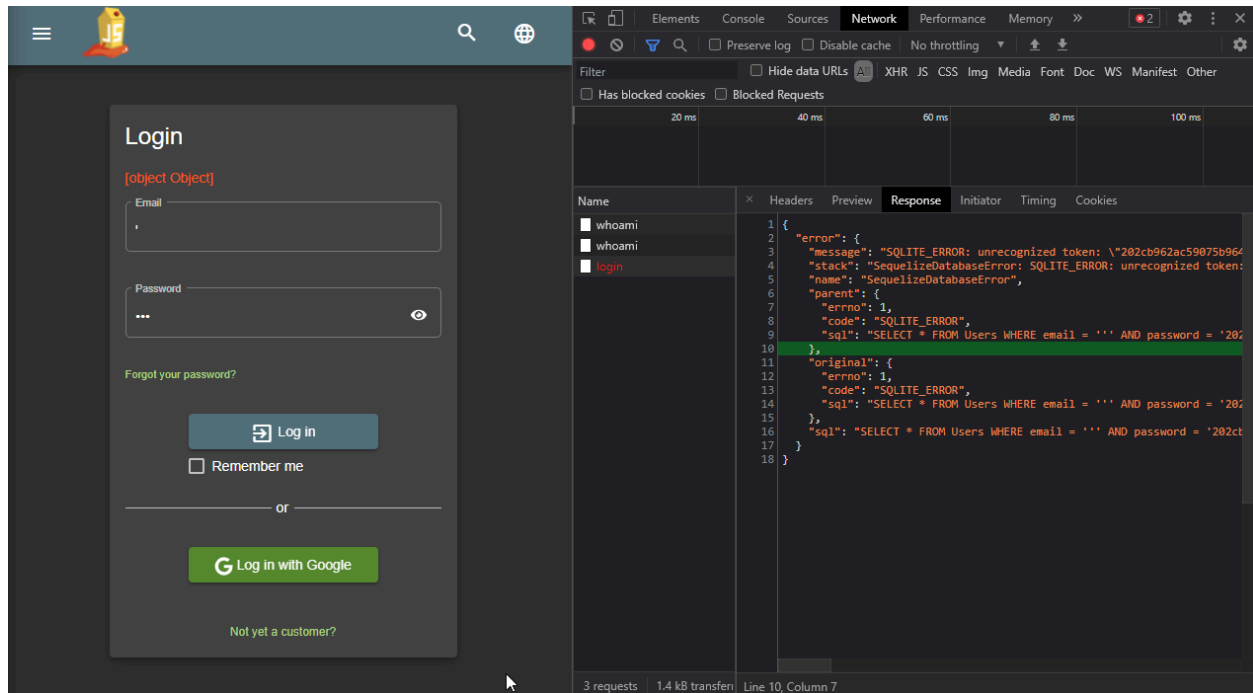
`<iframe src="javascript:alert(document.cookie)">`. For this next XSS were going to abuse the shops api and make a post request to the server with an XSS as the user.



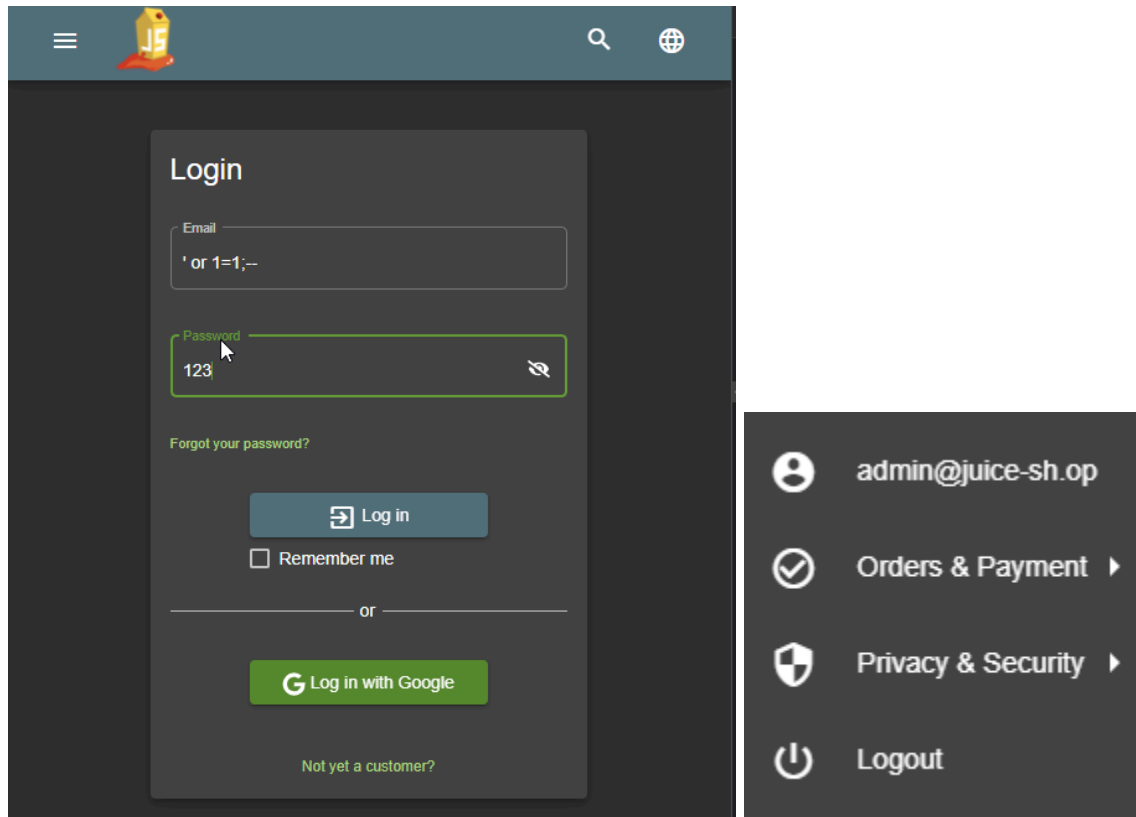
Here we have Insomnia, a tool used to help develop and test API's. With this we can trigger various different requests like POST, GET, PUT, DELETE and so on. In this specific example we want to post a working XSS snippet as a user without needing to go through the user facing account creation interface. By making this post request we do exactly that, causing an XSS to be in the administration page so long as it is registered as a user.



Lets begin to form some sql queries, the best place to start testing these queries out would be the login page.



Here we are going to start by placing a single quote in the first field and anything in the second we can test to see if we get [object Object] back from the server meaning that we are able to make invalid queries here. As well as this as you can see in the developer panel, we can look at the login response and look at the error provided to us. Not only is it not sanitizing SQL it's providing us with information as to what version of SQL the server is using to store data, in this case it's sqlite. Lets now try to form a query to log us into the admin account. Using some classic sql injection methods we can form something like this, ' or 1=1;--. At first glance this looks quite odd, this can work in older systems because the input gets placed into the SQL query that gets sent to the back end. Using the quotation mark in the first part of the query helps make it a proper query and the two hyphens on the end comments out the rest of whatever is being queried leaving 1=1 which is always true causing the password field to be true.



With that query we are now logged into the admin account. There are also other ways we can do this that we can also apply to any other account we may know the email to or a name within the email. To log into the admin account we can also use this query, ' or 1=1 and email like('%admin%');--. In this case this can work because the admin accounts email has admin within the name. So long as we know the name of an account on the site we can actually replace admin within the parameters with the name of the account like such, "' or 1=1 and email like('%bender%');--". Using this query we just constructed we can add the “not” statement before to make something like this “' or 1=1 and email not like('%admin%');--”, doing this one simple thing can actually log us into the account next stored in the database.

The image displays two screenshots from the OWASP Juice-Shop application. The left screenshot shows the 'User Profile' page, which includes a profile picture placeholder, an email field (jim@juice-sh.op), a username field (e.g. SuperUser) with a 'Set Username' button, and file upload options (Choose File, Upload Picture, Image URL). The right screenshot shows the 'Login' page, featuring an email field containing a malicious XSS payload (' or 1=1 and email not like('%admin%');--'), a password field (123), a 'Log in' button, a 'Remember me' checkbox, an 'or' separator, a 'Log in with Google' button, and a link for 'Not yet a customer?'.

Preventing a lot of these issues really has to do with the sanitization of inputs, using newer technology that helps prevent against XSS attacks like React which is a javascript framework for web development that actually on its own helps prevents many forms of XSS attacks and the proper authentication of API calls so that one can't just call POST requests from their machine and actually have data placed in the web server.

Conclusion

As it is OWASP Juice-Shop provides us with great insight into the standardization of web security. In terms of cyber security, there is no truly perfectly secure machine, so with standard practices and awareness we continuously work at an uphill battle just to mitigate the issues that we may not even know exist yet. Web security especially, is an extremely important and extremely vulnerable place. As practically everything is connected to the internet, the area of

web security is an ever expansive place with attack vectors and vulnerabilities truly everywhere. While we have all the tools to our disposal to cause damage on these sites, we can also use those very tools to protect them.

References:

- Q. Li, W. Li, J. Wang and M. Cheng, "A SQL Injection Detection Method Based on Adaptive Deep Forest," in *IEEE Access*, vol. 7, pp. 145385-145394, 2019, doi: 10.1109/ACCESS.2019.2944951.
- M. Agreindra Helmiawan, E. Firmansyah, I. Fadil, Y. Sofivan, F. Mahardika and A. Guntara, "Analysis of Web Security Using Open Web Application Security Project 10," *2020 8th International Conference on Cyber and IT Service Management (CITSM)*, Pangkal, Indonesia, 2020, pp. 1-5, doi: 10.1109/CITSM50537.2020.9268856.
- M. Dayal Ambedkar, N. S. Ambedkar and R. S. Raw, "A comprehensive inspection of cross site scripting attack," *2016 International Conference on Computing, Communication and Automation (ICCCA)*, Greater Noida, India, 2016, pp. 497-502, doi: 10.1109/CCAA.2016.7813770.
- OWASP juice shop. (n.d.). Retrieved April 02, 2021, from <https://owasp.org/www-project-juice-shop/>
- SQL injection prevention cheat Sheet. (n.d.). Retrieved April 02, 2021, from https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- <https://www.cloudflare.com/learning/security/what-is-web-application-security/>. (n.d.). Retrieved April 02, 2021, from <https://www.cloudflare.com/learning/security/what-is-web-application-security/>
- *OWASP Top Ten*. OWASP. (n.d.). <https://owasp.org/www-project-top-ten/>.