Ethical Student Hackers

Juice Shop

SHEFFIELD | Ethical Student Hackers

Breaking into security.

The Legal Bit

- The skills taught in these sessions allow identification and exploitation of security vulnerabilities in systems. We strive to give you a place to practice legally, and can point you to other places to practice. These skills should not be used on systems where you do not have explicit permission from the owner of the system. It is VERY easy to end up in breach of relevant laws, and we can accept no responsibility for anything you do with the skills learnt here.

- If we have reason to believe that you are utilising these skills against systems where you are not authorised you will be banned from our events, and if necessary the relevant authorities will be alerted.

- Remember, if you have any doubts as to if something is legal or authorised, just don't do it until you are able to confirm you are allowed to.

Code of Conduct

- Before proceeding past this point you must read and agree to our Code of Conduct - this is a requirement from the University for us to operate as a society.
- If you have any doubts or need anything clarified, please ask a member of the committee.
- Breaching the Code of Conduct = immediate ejection and further consequences.
- Code of Conduct can be found at https://shefesh.com/downloads/SESH%20Code%20of%20Conduct.pdf

SHEFFIELD Ethical Student Hackers
Breaking into security.

What is Juice Shop?

- Open Web Application Security Project (OWASP) Juice Shop is probably the most modern and sophisticated insecure web application!
- It can be used in security trainings, awareness demos, CTFs and as a guinea pig for security tools!
- Juice Shop encompasses vulnerabilities from the entire OWASP Top Ten along with many other security flaws found in real-world applications!
- It’s basically a very insecure application that can be exploited in multiple ways in order to gain an understanding of possible exploit vectors that are common in some websites.

How to connect to the website

https://juice-shop.herokuapp.com/#/

Or simply Google ‘OWASP Juice shop’

Using TryHackMe (If you have the setup for it)

1. Create a TryHackMe account
   a. https://tryhackme.com/
2. Download the OpenVPN file
   a. tryhackme.com/access
3. Connect to the OpenVPN server
   a. sudo openvpn [Path To File]
   b. Equally on windows you can download the OpenVPN software from https://openvpn.net/client-connect-vpn-for-windows/
4. Join the TryHackMe Juice Shop room and start an instance
   a. https://tryhackme.com/room/juiceshop

Exploring the Website

- Alright! Now that we're (hopefully) connected, let's have a look around the website!
- There are multiple sections to this website that look interesting at a first look
  - Login form - Default creds? SQL injection?
  - Contact form - Very simple looking captcha
  - /score-board - Depending if you're using the online or TryHackMe
  - About Us - Contains some links
  - The home screen

The Score-Board Page

Let's look around

- Looking at the source of the index page, we can see that in main.js there is a link to some pages that are not visible when you navigate the pages (We first have to beautify it, then the pages should be at the bottom).
- This exposes /administration and /score-board
- Having a look at these we can see that there is an administration panel and also a score board.
- We didn’t have to be logged in as a user to see this information, it was given to us when we first loaded up the website.

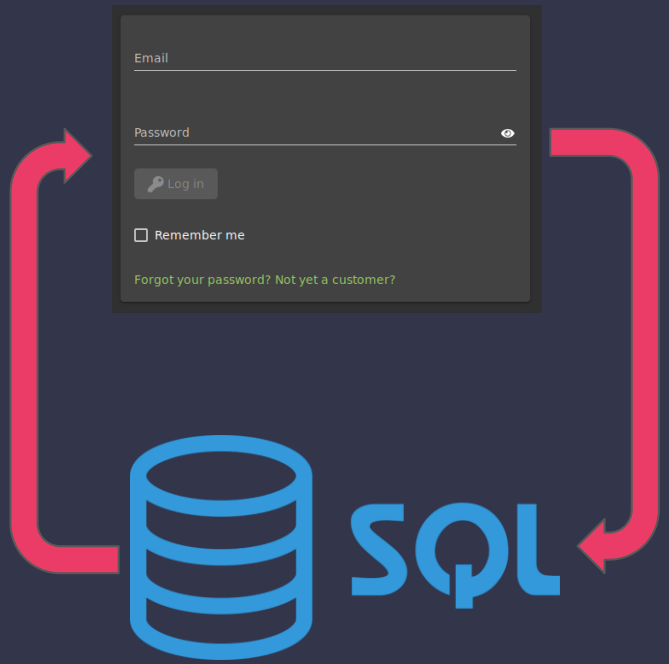
How do we find the page?

- Let's have a look at the javascript file that is run on the website
- CTRL + SHIFT + I
  - Opens developer console for Chrome and Firefox
- Head over to the debugger and look at some of the .js files
  - You may need to beautify the javascript to make it a bit more readable
  - https://beautifier.io/
- At the bottom of the page we can see some interesting strings
- Now we can see the different type of vulnerabilities we can find on the website, as well as some other information such as the administration panel.
- Also inspect on the online website shows the score-board link is hidden

The Login Form

Taking a look at the login form

- Looks like a fairly simple login form
- This website links to an SQL database in the back of the website
- This means that, at some point, a query is sent to the database that asks the database if there is a valid user given the username and the password.
- A default query for this would be:
  - SELECT * FROM users WHERE username = '[username]' AND password = '[password]'
- If some data gets returned by this, then there is a valid user in the database, if not then the user cannot log in.
- We can exploit this if it has not been implemented correctly using SQL injection!



SQL Injection

- First of all, let's see if the login form is vulnerable - to do this, we'll try to provoke an error by submitting a quotation mark (“ or ‘)
- The site returns an SQL error! This is because SQL (the language for talking to databases) uses quotation marks as special characters to ‘wrap’ strings - adding an extra one breaks the pattern!
- Looking again at the statement from the last slide, we can guess how the database queries its data
  - SELECT * FROM users WHERE username = ‘[username]’ AND password = ‘[password]’
- If we enter in some SQL code, such as “ ‘ or 1=1;-- “ then we could possibly login to the website.
  - This works out, in the database's context, as:
  - SELECT * FROM users WHERE username = “ or 1=1;-- ’ AND password = ‘[password]’
  - The red coloured text is the SQL that is actually run, the grey has all been commented out by the -- and is therefore ignored

Logging in as other users

- So we found we can login to the website using some very basic SQL injection as an admin account
  ‘ or 1=1;--

- However, we can also login to other user accounts too. Given that the previous SQL statement will log us into the first account in the database, we can refine the statement down so that is selects a specific account.

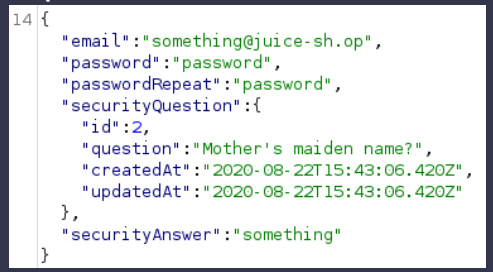
- ' or 1=1 and email like('%jim%');--
  This allows us to login to the user account jim, however we can also find other accounts on the /administration page that we can also login as

Looking at the user token

- First, let's have a look around at the cookies that are set when we create an account
- We can see that some form of token is created and set as a cookie. This can be decoded using https://jwt.io, an online tool for decoding json web tokens.
- Once we have decoded the token, we are able so see some information about the user we have logged in as. Some of this information can be very useful to us, for example the id of the user, the email, password, as well as if the account is an admin or not.
- Given that we can now login to any of the user accounts using the SQL injection, we are now able to obtain the password hashes of the user accounts.
- However this also gives us some more information on the backend of the server, that there is some form of isAdmin value.

Creating admin accounts

- Running based on the knowledge we learnt previously, we can see that there is an isAdmin variable that is either true or false.
- Now let's have a look at what happens when we create a new account in burp, we can see that there is a email, password, passwordRepeat and the security question.
- However we know that there is an isAdmin field that's in the user token
- If we intercept the request to create the account and add the role field and set it to "admin", will it create an admin account instead of it being set to a false by default and creating a normal account?



Finding information about Jim

- Using two of the previous exploits we found, we can start off by logging into the Jim users account, and then find out the hash of the password using the token exposure.
- We can then either use John or https://md5decrypt.net/en/ to brute force (or rainbow table) the password. This gives us a password to login to his user account.
- Now, going off what the password is, and what the security question is, we can make an educated guess at what is security question reset is.
- Let's do some digging…
- https://en.wikipedia.org/wiki/USS_Enterprise_(NCC-1701)
    - James T. Kirk's Brother's middle name (As per the security question)

The About Page

FTP Access through /about

- Having a look around the about page, we can see a rather long hyperlink in the middle of the lorem ipsum text
- This hyperlink takes us to an interesting directory of the website as it seems to be hosting FTP
- Navigating to the root of this directory we can see a load of exposed files that we probably shouldn't be able to see
- If we didn't have this list, we could use something like wfuzz or gobuster to discover the files
- Not all of the files are readable, however. Only the ones ending in .pdf or .md
- We can bypass this by using a method called null-byte injection

Null byte injection

- A null byte is a control character that has the value of 0. It is used to terminate the end of a string in most programming languages
- It can be represented in many ways throughout different contexts, \000, \x00, \z, \u0000 or %00
- For our exploit, we want the server to think our file ends in .pdf or .md but for it to actually read a file that may not have that extension
- If we try accessing the directory /ftp/coupons_2013.md.bak%00.md then we get a bad request.
  - This is because this website, for some reason, is decoding the URL twice, therefore we need to account for this
  - We need to URL encode the percent, that will then allow us to URL encode the null byte (A little confusing I know)
  - /ftp/coupons_2013.md.bak%2500.md - %25 == % in URL encoding

Decoding the coupons

- All of the coupons seem to look very similar, with only a couple of changes towards the beginning of the string
- This could imply that there is some form of encoding being done on the string. (Equally it could just be the format of the coupons)
- Looking into the package.json.bak file we can see a list of the libraries that are being used on the website. Among the list is a package called “Z85” which is an encoding format
- Therefore if we decode the string using z85 we can see the original string. It’s in the format JAN15-10, meaning the MonthYear-Discount. We could then try a load of our own encoded coupons to see if we can get a better discount
    - JUL20-99 - n(XLufFbpB - Works when I made this slide

XSS Injection

What is XSS Injection?

- XSS is an abbreviation of cross-site scripting
- This is a vulnerability where a user is able to inject client-side scripts (typically javascript) into a webpage for it to be loaded by other users when they access the page on the website
- XSS can come in three different forms:
    - Persistent (stored) - The XSS is stored on the servers, such as in a database, and is delivered back to the user whenever they access the website, for example in the form of a comment section of a website.
    - Reflected - The XSS is delivered as part of the HTTP request, for example as a parameter in a search field. When the user visits the given URL, the malicious script is loaded.
    - DOM - The XSS is delivered via the website's Document Object Model (i.e. its elements), for example by modifying an element via a script built into the site.
- https://owasp.org/www-community/attacks/xss/

How can we use XSS on Juice Shop?

- There are a couple of fields that can be used to exploit XSS on the website, both in reflected and also stored forms. Any ideas?

- Heading over to the search box on the home screen, you can see that the search query is rendered back onto the webpage when you search for something. This means that it may be possible to inject some malicious code and for it to be run on the clients browser.

- Using the iframe code that is given to us on the score-board page, we can cause an alert to appear on the page.

- There are a couple more places on the website where an XSS can be performed, such as the customer feedback form (whose results are displayed on the about page). This is a bit tricker to perform though, as it has better sanitisation. We cover this in the Give it a Go worksheet

What about something less harmless?

- Stealing a cookie and displaying it to the page
  - Use document.cookie and the alert method from earlier
  - <iframe src="javascript:alert(document.cookie)">

- Redirecting to another URL
  - Usually we'd use window.location.href('url');
  - If we're going from an iframe, we need to use window.top.location.href('url');
  - For example, <iframe src="javascript:window.top.location.href = 'http://www.shefesh.com';">

Manipulating the baskets

Looking into other peoples baskets

- There is a lack of authentication with the baskets in the website. We can look into other users baskets without having to be logged into their accounts.
- To do this, we simply need to look at the requests that we make to the website in order to see the contents of the basket. To do this we will open burp suite.
- Simply navigate to the basket page of the website while intercepting the packets, you should be able to intercept a request to /rest/basket/[basketID], where the basketId is the users ID
- If we simply change the baskets ID then we should be able to get a JSON response of the contents of the basket.

Basket Manipulation

- Now, if we head over to the #/basket section of the website, we can see that we are able to “purchase” the items that we have added to our own basket.
- Let’s start of by intercepting the traffic when we add an item to the basket
- We can see that there are 3 parameters that are in the post request, the itemId, the basketId and the quantity
- We can play around with these values a little bit to see what we can affect
  - If we play with the basket number we can choose who’s basket we place the goods in.
  - If we play with the item id we can select items that may not be displayed on the purchase screen.

Free Money!

- In the last slide we could see that we were able to edit the quantity of items in our basket based on the requests that we made

- Taking this further, we are able to actually add money to our account using this lack of authentication. This is due to the simple validation and maths behind the application.

- If we where to visualise this, it could look like amountToPay = costOfItem * quantity. So if the quantity is negative, the amount to pay ends up negative. This is then used to update the balance: balance = balance - amountToPay. However the amountToPay is negative, so we are adding money to the balance.

- This may only be visible in the newer version of Juice Shop (The one hosted online), as the Juice Shop on TryHackMe is a little bit older. TryHackMe JS can still be vulnerable to the negative balance though.

Juicy Extras

Posting Reviews

- Some more lack of validation can be exploited with the reviews system too!
- It's getting a little repetitive now, but we'll be using burp suite again to intercept and inspect the requests made to the application. It just shows how useful Burp can be!
- Let's post a review on one of the products, and inspect the request that is made.
- We can see that there are two parameters when posting a review, the message and the author (We can also edit the product we review in the url bar)
- Now we are able to impersonate reviews for other users, as we are able to arbitrarily set the email and also the message.
- We can even edit reviews of other users! You can try this in the worksheet

Creating our own valid JWT Token

- We are able to login to any account that we want to, even if it's not a valid account to login as!
- This is done via the SQL injection that we did at the beginning, we are able to artificially return some values from the 'database' so that the backend can create a valid JWT token for us.
- This is done by using the UNION statement in SQL, it basically appends some information on to the end of the previous statement as long as the column numbers are the same.
- ' UNION SELECT * FROM (SELECT 200 as 'id', 'chad' as 'username', 'chadd@juice-sh.op' as 'email', 'chadd' as 'password', 'admin' as 'role', '' as 'deluxeToken' , '1.3.3.7' as 'lastLoginIp' , 'assets/public/images/uploads/default.svg' as 'profileImage', '' as 'totpSecret', 1 as 'isActive', '1999-08-16 14:14:41.644 +00:00' as 'createdAt', '1999-08-16 14:33:41.930 +00:00' as 'updatedAt', null as 'deletedAt')--

Extracting the Database Schema

- Search queries are directed to /rest/products/search?q=
- Experiment to find that the original query is closed by ‘))
- Now let’s try a UNION SELECT !
  - UNION SELECT joins up a second query from another table
  - What table should we join up? In SQLite (which we know is the engine from error messages), schema is stored in sqlite_master
  - So let’s try ')) UNION SELECT * FROM sqlite_master--
  - Now we need to match up the columns - a requirement of UNION SELECT
  - Keep adding arbitrary values: ')) UNION SELECT '1' FROM sqlite_master--, ')) UNION SELECT '1', '2' FROM sqlite_master--, …
  - Eventually, we find ')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM sqlite_master-- works!
  - Now select ‘sql’ as a column, and provide a search term to filter out unwanted results: ethical_hackers')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM sqlite_master--
- Now we can read the CREATE statements used for each table!
- Note: the column names will still be written in terms of the Products table

Adapting this Query

- Now we have a URL that we can use to grab the schema from the database:
  - https://juice-shop.herokuapp.com/rest/products/search?q=qwert%27))%20UNION%20SELECT%20sql,%20%272%27,%20%273%27,%20%274%27,%20%275%27,%20%276%27,%20%277%27,%20%278%27,%20%279%27%20FROM%20sqlite_master--
  - (where %20 and %27 represent an encoded space or apostrophe)
- We don't have to stop here!
  - We can change the table name to select from a different part of the database
  - We can use the query above to find out the column names for that table, and select them instead
  - Now you can steal any data you like! There are some suggestions on the worksheet

That's It!

Any Questions?

Thirsty for more?

- Take a look at the worksheet from the GIAG
- Have a go at this week's worksheet!
- Take a look at the rest of the scoreboard - you should be able to have a go at some of the harder challenges now!
- Join the Juice Shop channel on our discord!

Upcoming Sessions

What's up next?

www.shefesh.com/sessions

Guest Session - YHROCU (EGM meeting at 5:30PM)

Automation in Python

Networking

All the Shells!

Any Questions?

www.shefesh.com

Thanks for coming!

SHEFFIELD Ethical Student Hackers

Breaking into security.