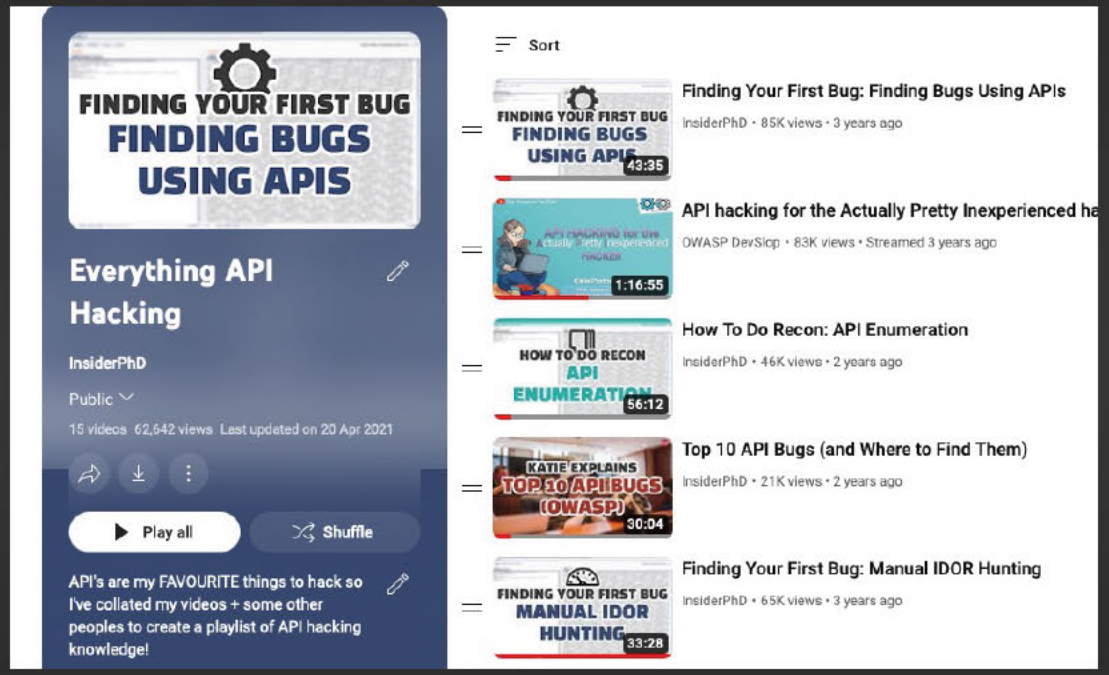
PRESENTATION

How to Properly Own API's for Your First Valid Submission

SATURDAY, JUNE 17TH
@INSIDERPHD

NAHAMCON.COM

About



- Katie aka InsiderPhD
  - Hacker and YouTuber
  - Specialise in APIs
  - Used to make them now I break them
- Tons of videos on API hacking if you want to learn more
- This is a short talk to inspire you to try out API hacking and talk about some of the bugs you should look for!
- I have demos, tools, more bugs on YouTube
  - but these are 30mins – 1hr long!

Why APIs?

Very common design for modern web apps and mobile apps

New technology like GraphQL is changing how APIs work

Less familiarity with APIs vs web applications

Large Attack Surface

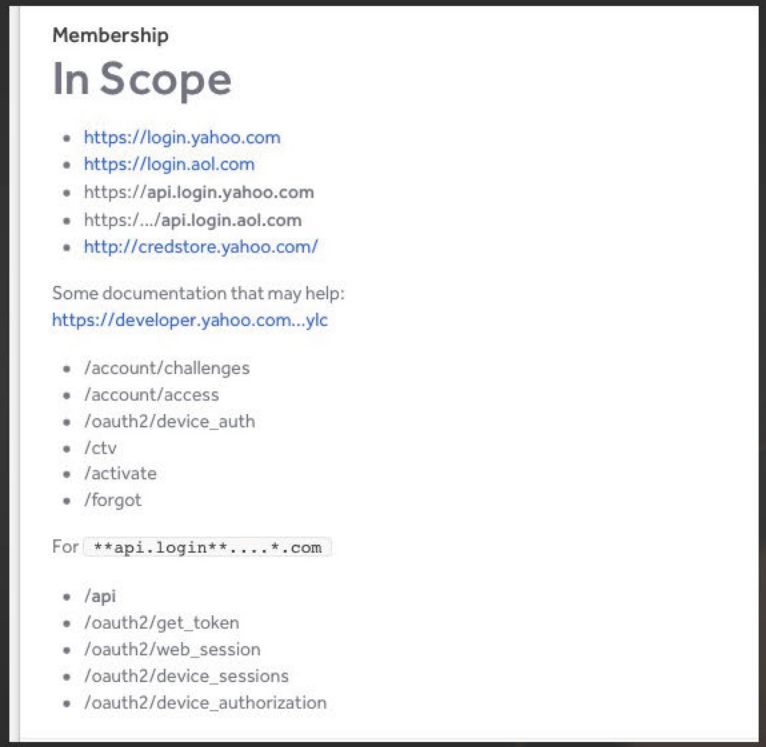
Often generated rather than being designed

Intimidating for other hackers

Client side fixes, but not server side

Pick the right target

- Large API attack surface but not enterprise
- Yahoo - lots of functionality all use APIs
- Tumblr – big monolithic API
- Don’t attack something with a small surface area
  - You need to find niche features
- Mobile app + Desktop app



Membership

In Scope

- https://login.yahoo.com
- https://login.aol.com
- https://**api.login.yahoo.com**
- https:/.../**api.login.aol.com**
- http://credstore.yahoo.com/

Some documentation that may help:
https://developer.yahoo.com...ylc

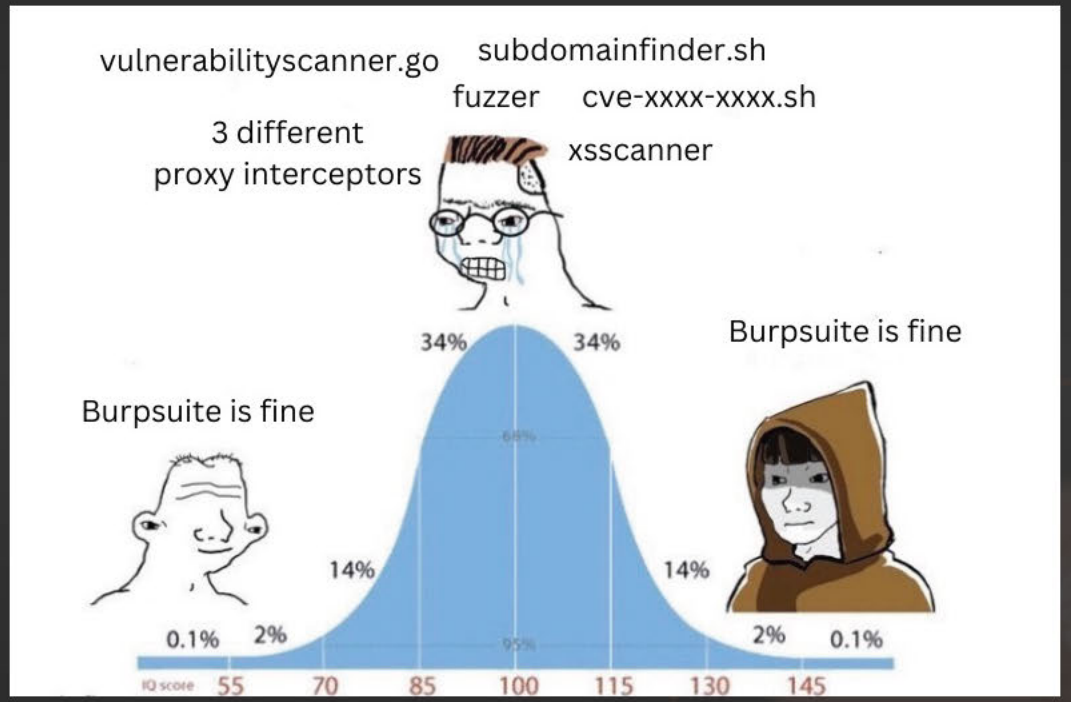
- /account/challenges
- /account/access
- /oauth2/device_auth
- /ctv
- /activate
- /forgot

For `**api.login**....*.com`

- **/api**
- /oauth2/get_token
- /oauth2/web_session
- /oauth2/device_sessions
- /oauth2/device_authorization

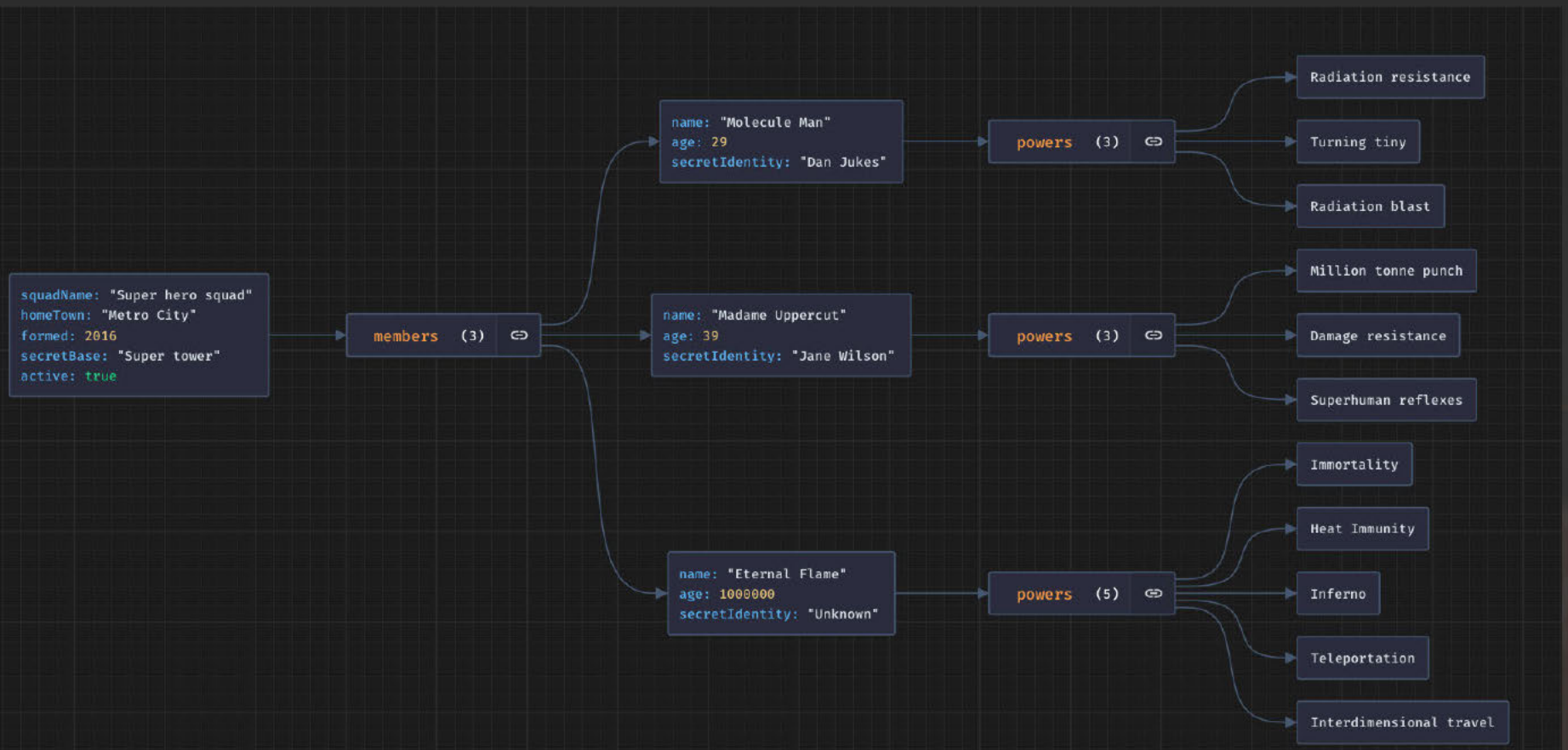
Recon

- Don’t overcomplicate it
- Try and find rarely used features in the app
  - Go into hamburger menus
  - Do unexpected things
  - Pay money for more access – depends on how expensive
- You don’t need anything other than Burp!

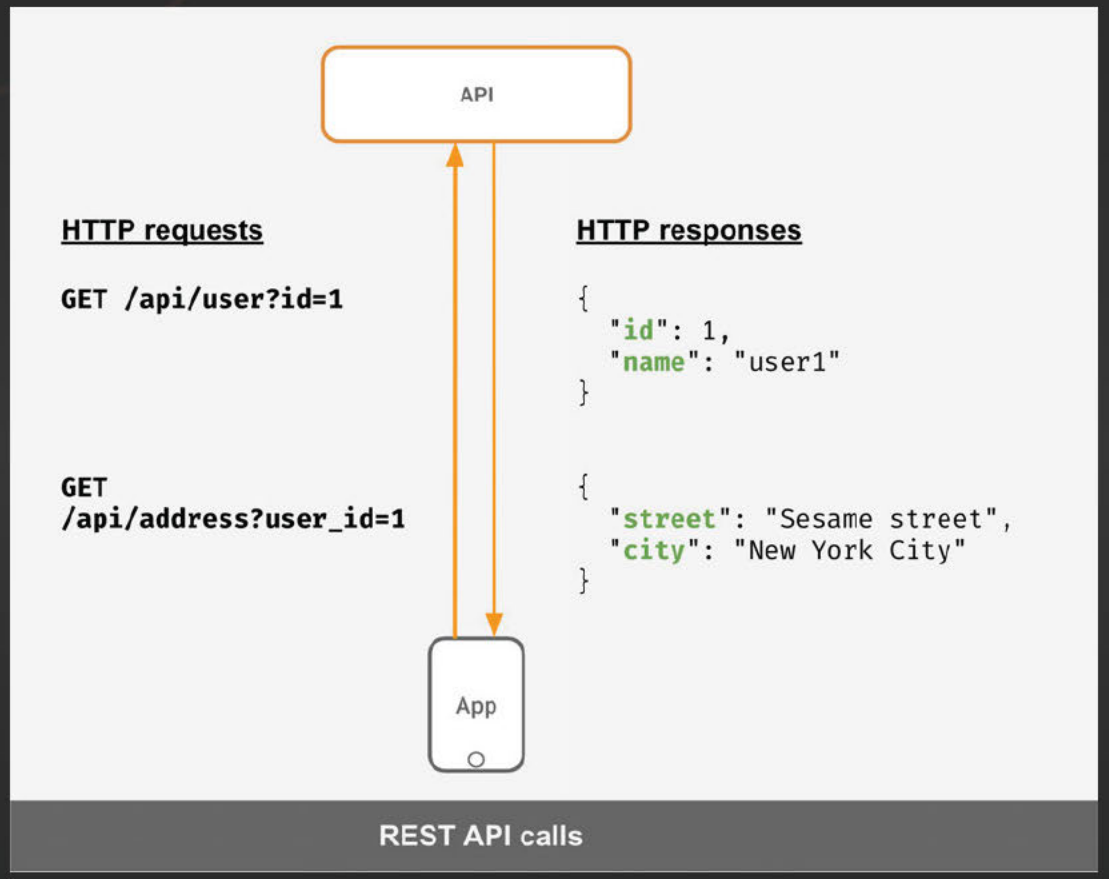


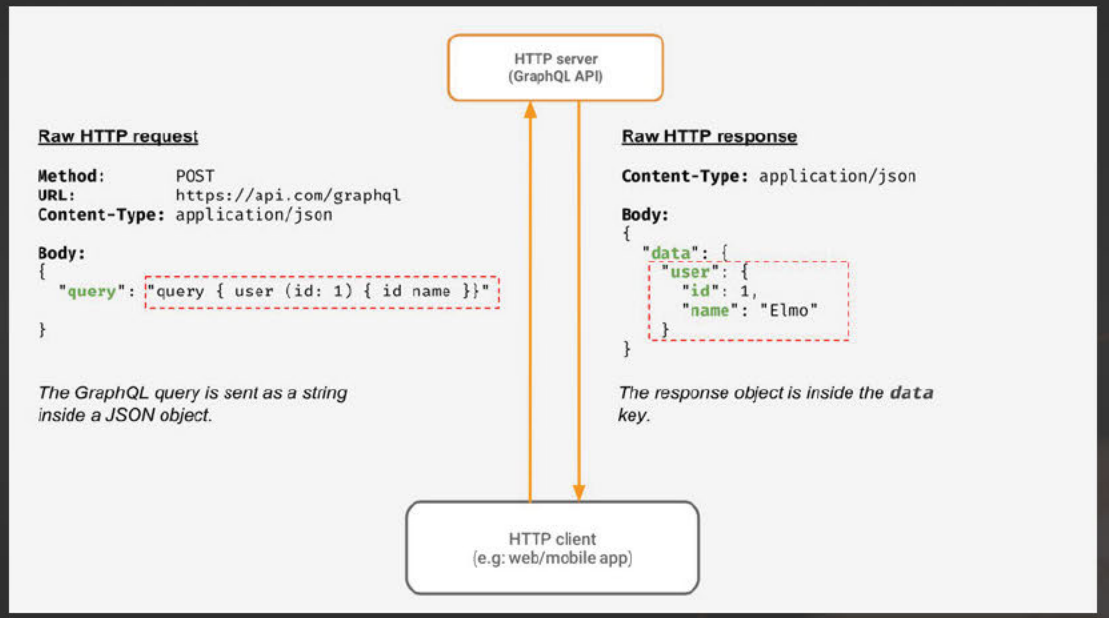
Read JSON

```json
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```



Making requests





Notes will help you make sense of a complex API

	A	B	C	D	E	F	G	H
1	Method	API endpoint	Broken Object Level Authorisation	Information Disclosure	Broken Function Level Authorisation	Priviledge Escalation	Business Logic Issues	Needs Authenticati on?
2	GET	/api/buildings		X				
3	GET	/api/buildings/1		X				
4	POST	/api/buildings			X			
5	PUT	/api/buildings/1	X					
6	DELETE	/api/buildings/1	X					

Issues to test for

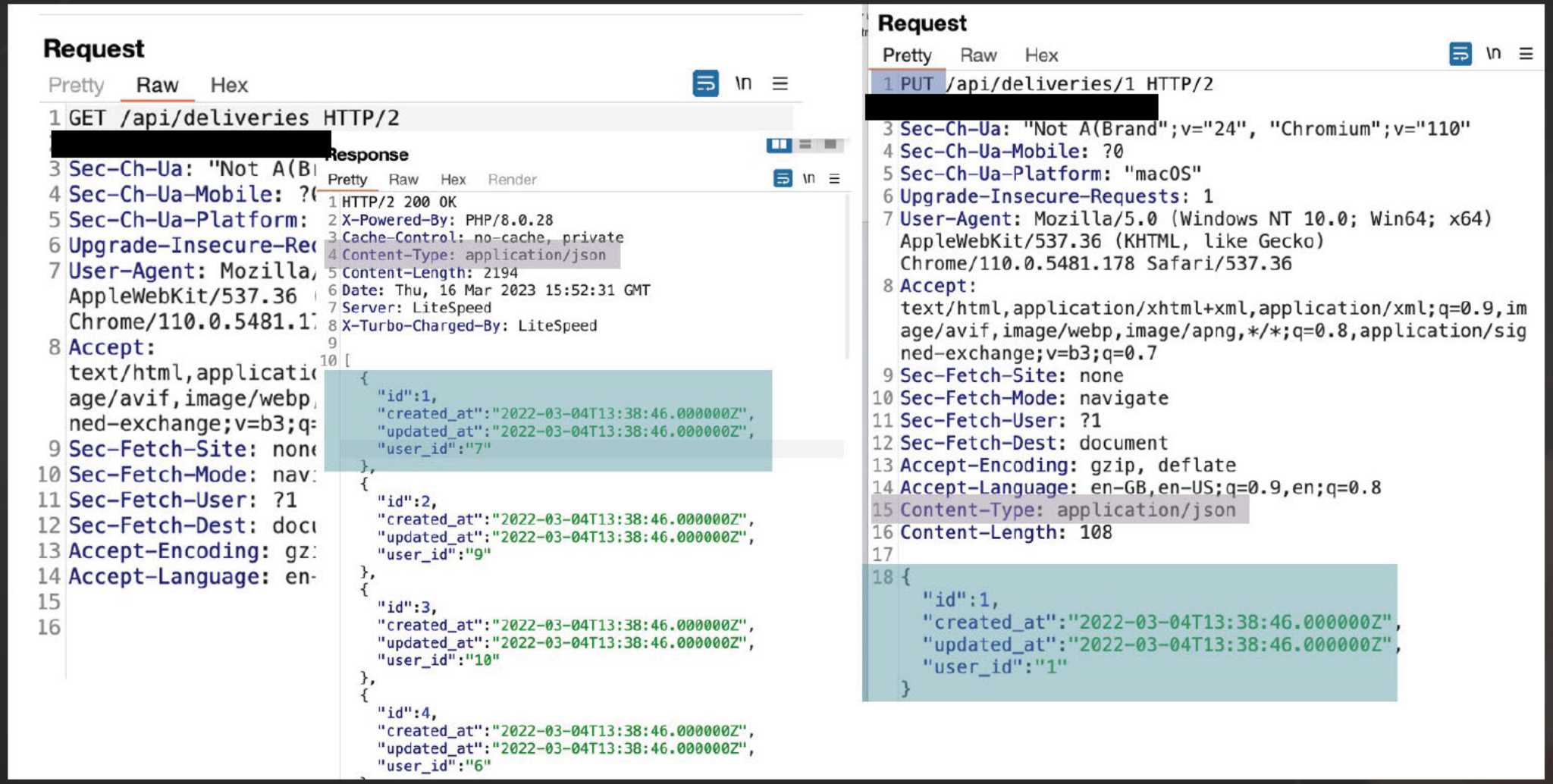
Access Control – IDORs	• Cross user, Permission issues, Cross tenant
Mass Assignment	• Test a wider range of inputs
Information Disclosure	• Read the response, is it sensitive or user data?
Business Logic Errors	• Very common especially with complex permission hierarchies

Access Control

- Make 3 accounts – admin, regular user 1 , regular user 2 (and guest if that's separate)
  - Note which requests/functions need what permission levels – use a spreadsheet if required
  - Check and see if you can repeat the request for an admin with a regular user account – swap the cookies
- Test user/tenant access control in the same way, can user 2 repeat user 1s request and still affect user 1?
  - If testing tenants make sure you have more users in different tenants

		Permissions							
		edit profile	search posts	create post	edit post	update post	delete post	edit basic users	edit all users
Level		1000		2000			3000		4000
Permission Level Number		1001	1002	2001	2002	2003	3001	3002	4001
Role	Basic User								
	Paid User								
	Limited Administrator								
	Full Administrator								

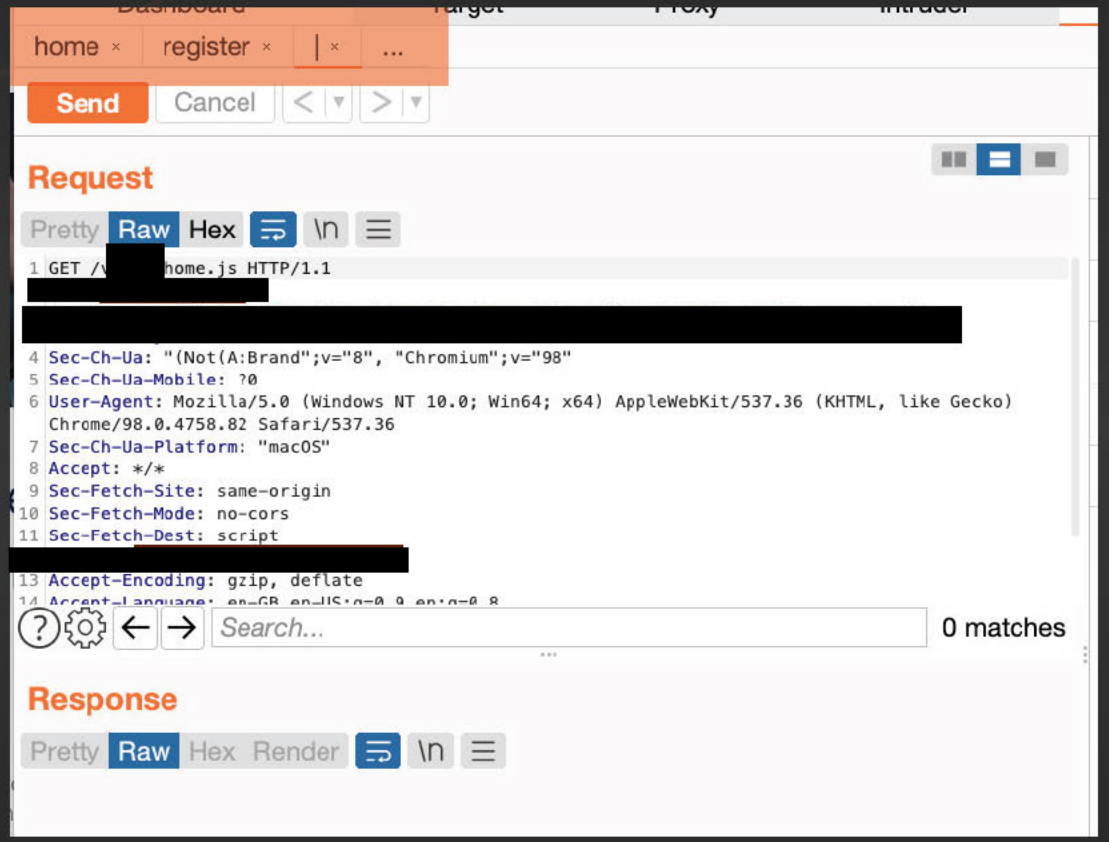
Mass Assignment



Information Disclosure

- READ THE JSON
- IS IT SENSITIVE? And what is the impact of it?
- IF YES REPORT IT
- Become familiar with things like GDPR to make an argument for personal data!

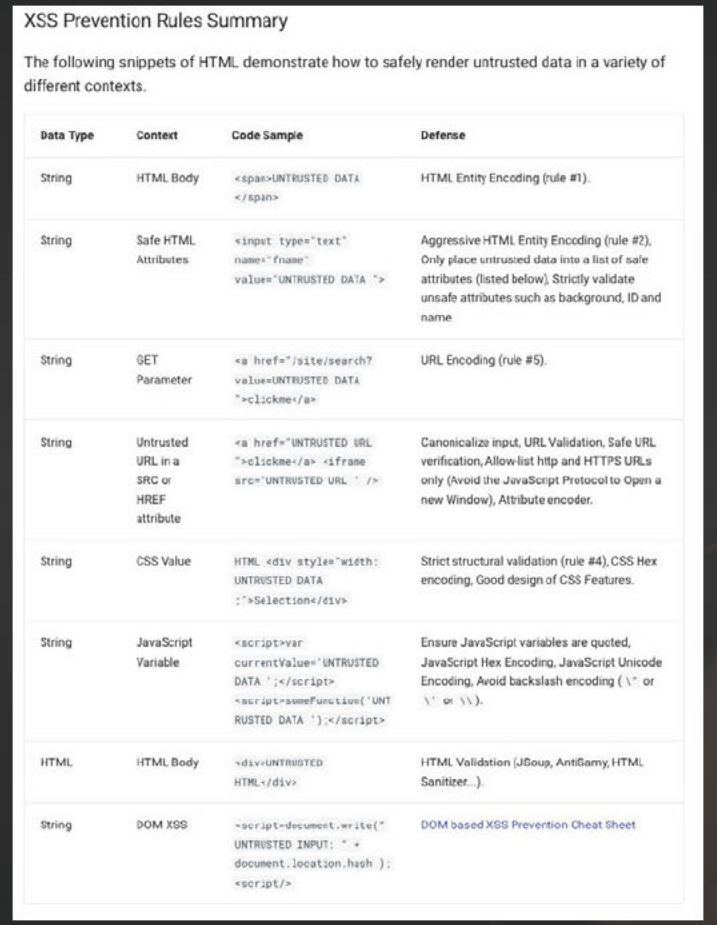
Business Logic Errors



- Walk through features of the API, try and skip over steps
  - E.g. skip the payment step
- Or try and put resources in odd states
  - e.g. having an order that is cancelled but out for delivery
- Or try and place conflicting access control rules
  - E.g. a guest user that can edit their email but can't read their user details

Input Sanitisation Issues

- Really common on mobile apps
- Check for the usual suspects e.g. XSS
- Use the update request to add the XSS payload
- Then view the payload on the website directly
- If you see < on the page it's not vulnerable



XSS Prevention Rules Summary

The following snippets of HTML demonstrate how to safely render untrusted data in a variety of different contexts.

Data Type	Context	Code Sample	Defense
String	HTML Body	`<span>UNTRUSTED DATA </span>`	HTML Entity Encoding (rule #1).
String	Safe HTML Attributes	`<input type="text" name="fname" value="UNTRUSTED DATA ">`	Aggressive HTML Entity Encoding (rule #2). Only place untrusted data into a list of safe attributes (listed below). Strictly validate unsafe attributes such as background, ID and name
String	GET Parameter	`<a href="/site/search?value=UNTRUSTED DATA ">clickme</a>`	URL Encoding (rule #5).
String	Untrusted URL in a SRC or HREF attribute	`<a href="UNTRUSTED URL ">clickme</a> <iframe src="UNTRUSTED URL " />`	Canonicalize input, URL Validation, Safe URL verification, Allow-list http and HTTPS URLs only (Avoid the JavaScript Protocol to Open a new Window), Attribute encoder.
String	CSS Value	`HTML <div style="width: UNTRUSTED DATA ;">Selection</div>`	Strict structural validation (rule #4), CSS Hex encoding, Good design of CSS Features.
String	JavaScript Variable	`<script>var currentValue='UNTRUSTED DATA ';</script> <script>someFunction('UNTRUSTED DATA ');</script>`	Ensure JavaScript variables are quoted, JavaScript Hex Encoding, JavaScript Unicode Encoding, Avoid backslash encoding (`\"` or `\'` or `\\`).
HTML	HTML Body	`<div>UNTRUSTED HTML</div>`	HTML Validation (JSoup, AntiSamy, HTML Sanitizer...).
String	DOM XSS	`<script>document.write(" UNTRUSTED INPUT: " + document.location.hash ); <script/>`	DOM based XSS Prevention Cheat Sheet

Advice

- API hacking is much more about volume than skills
- APIs take time to hack manually – it's why they're good for beginners
- You are going to find dupes – THAT'S OKAY dupes mean you were GOOD ENOUGH you just weren't FAST ENOUGH
  - I've got a ton of dupes, so do most other hackers!
- Try and focus on doing things manually rather than automation
  - Someone else has probably already automated it and has scaled it
  - Automation will likely miss things, even semi automation like autorize
- Complex APIs are your friends – yes they are harder to learn and use but less opportunity for dupes!

Thanks for listening ☺

You can follow me @insiderphd on twitter or on youtube