Hacking and Defending APIs

Red and Blue make Purple

TABLE OF CONTENTS

01 **Intro**
Quick background and such

02 **Why Attacking APIs?**
What makes APIs interesting to attackers

03 **Attacking APIs**
How to attack and what those attacks look like

04 **Conclusion**
Key takeaways and your questions

Who is this guy?

- Reformed programmer & AppSec Engineer
- Noname Security -
  Distinguished Engineer, Noname Labs
- 15 years in the OWASP community
  - OWASP DefectDojo (core maintainer)
  - OWASP AppSec Pipeline (co-leader)
  - OWASP WTE (leader)
- 22+ years using FLOSS and Linux
- Currently a Go language fanboy
- Ee Dan in Tang Soo Do Mi Guk Kwan
  (2nd degree black belt)
- Founder 10Security

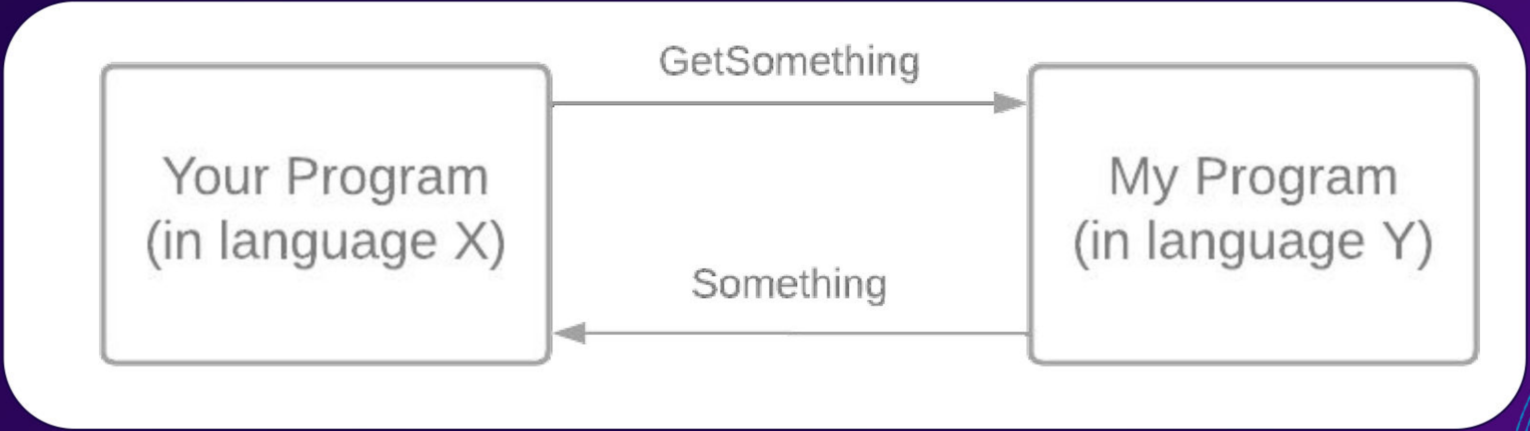
02

Why attack APIs?

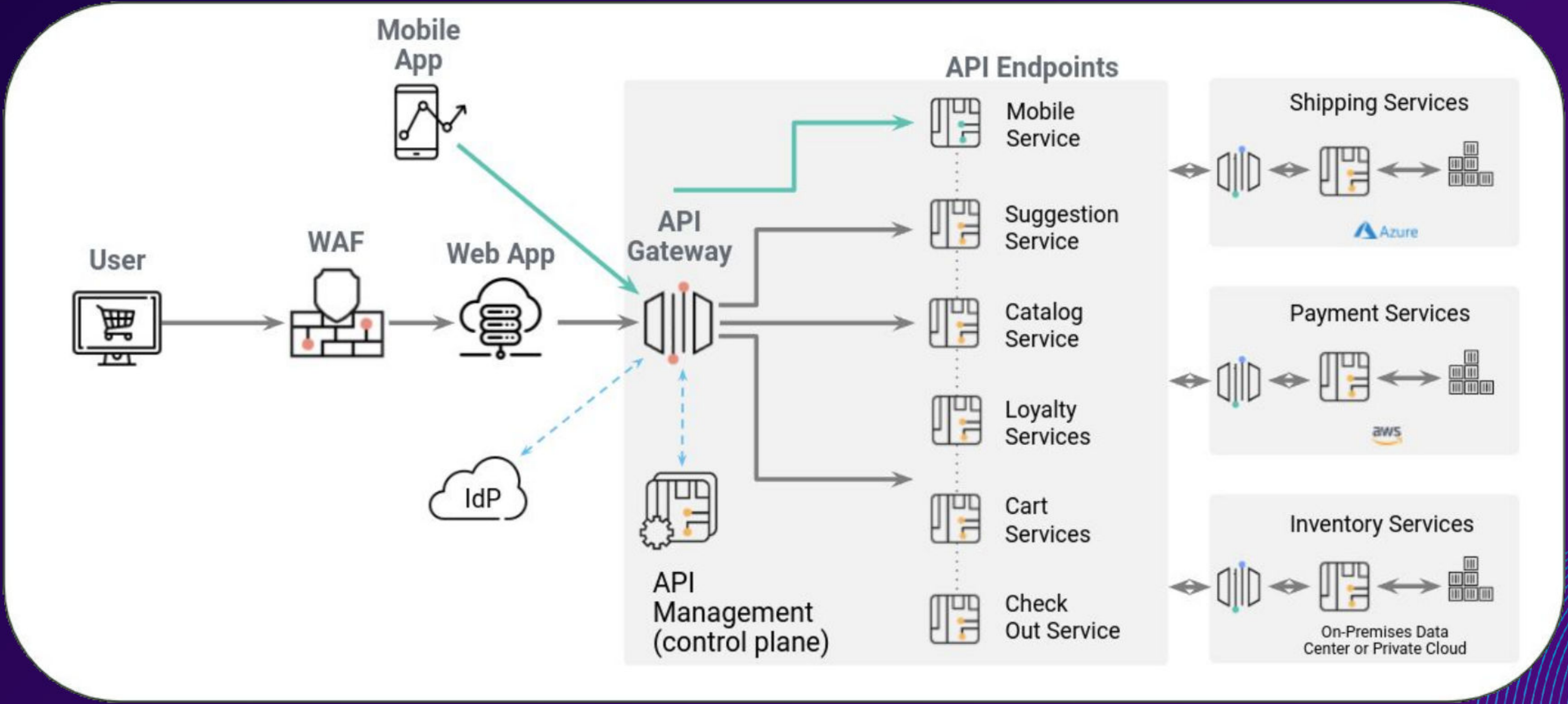
APIs are Simple

Wikipedia:

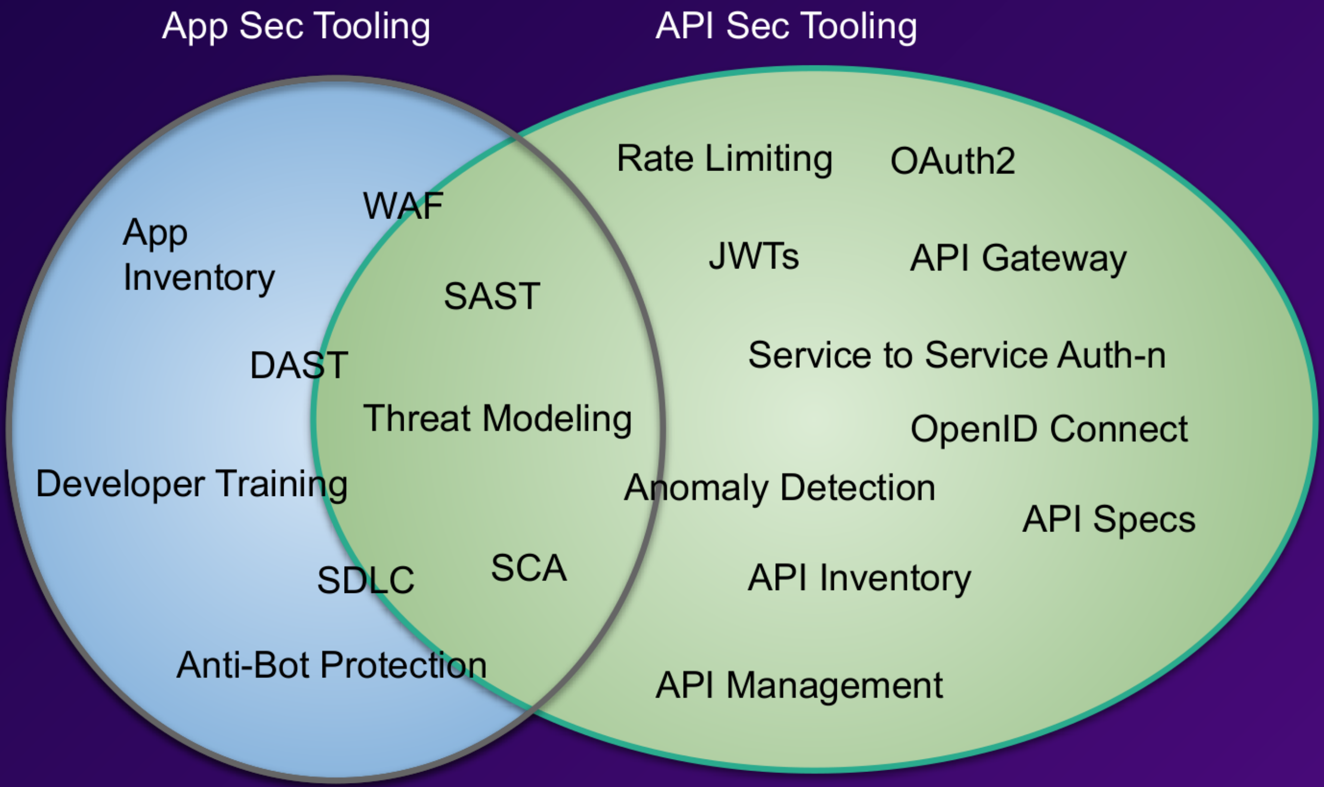
An application programming interface (API) is a connection between computers or between computer programs.



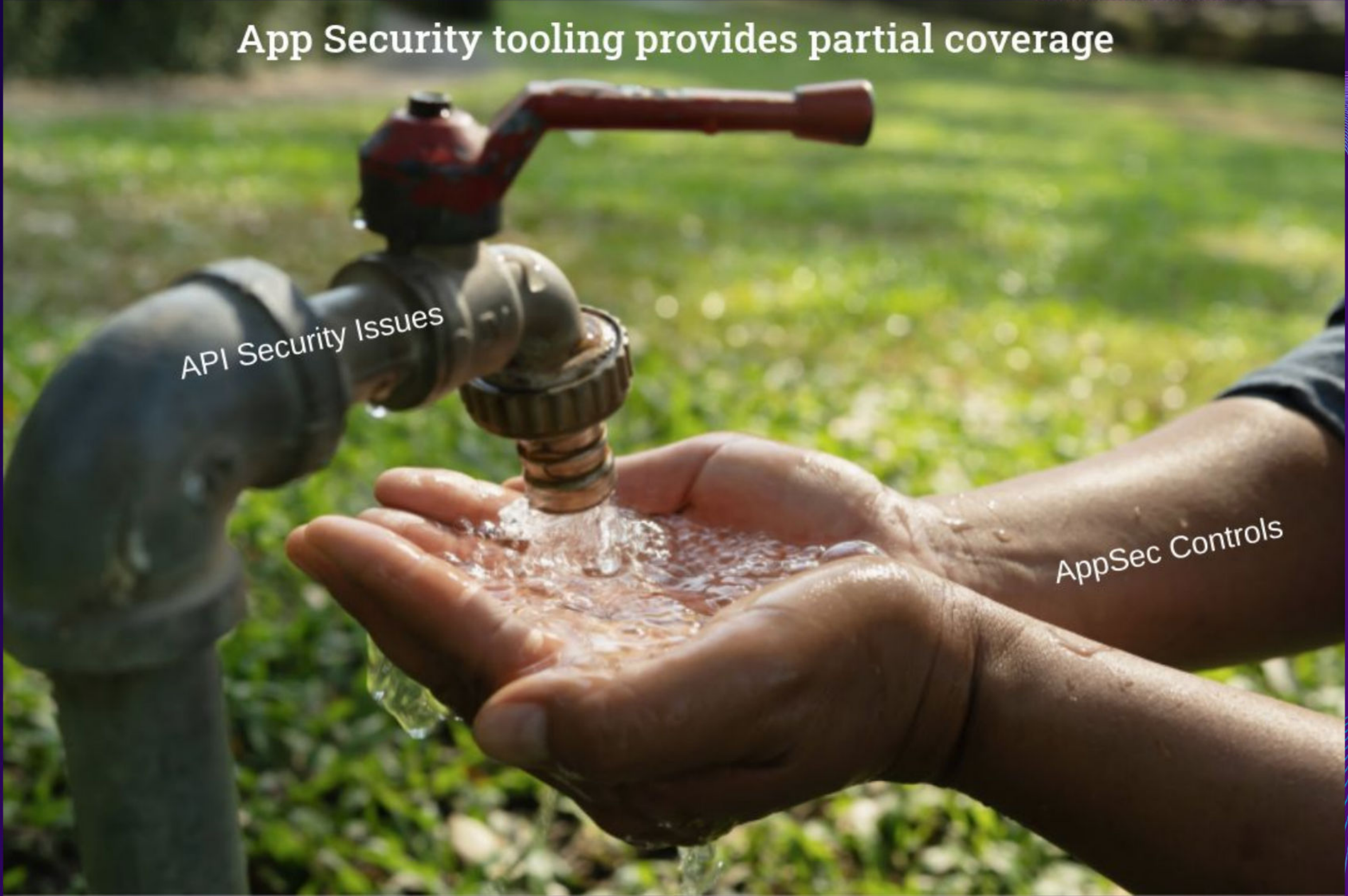
APIs aren't Simple



Even if you have a solid AppSec program



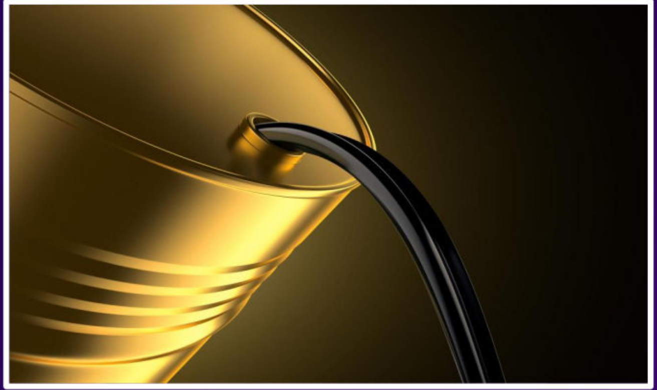
App Security tooling provides partial coverage



It’s All About the Data

“Data is the new oil”

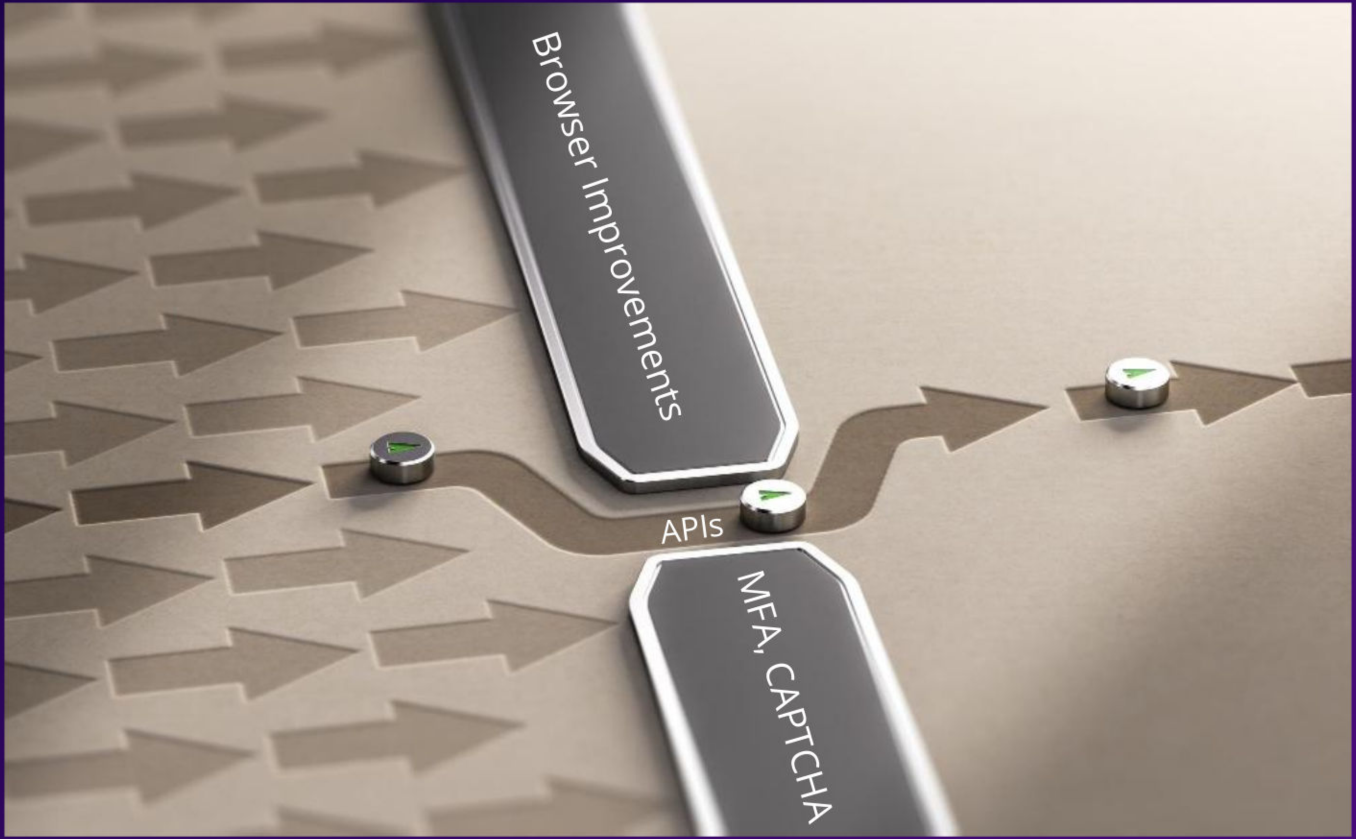
Clive Humby
British Mathematician



“APIs are data pipelines”

Matt Tesauro
Your presenter

As browsers and web apps get hardened…



Types of API ~~attacks~~ Testing

Blackbox

Simulate an attacker
with zero knowledge

Whitebox

Test with full knowledge,
some controls turned off

Greybox

Like blackbox but with
limited info on the target

Crystalbox

Full knowledge including
source code, only the
APIs controls in place

Pro Bono Pen Testing *(attacks)*

Defining the 3 Pillars of API Security

1. **API Security Posture**
   a. Full inventory of all APIs
   b. Who is calling the API? What data is sent/received? Where did the call originate?

2. **API Runtime Security**
   a. Watching API traffic and understanding what is normal
   b. Anomaly detection and alerting

3. **API Security Testing**
   a. Assess the security state of APIs
   b. DAST, not SAST ideally tested early and often
   c. Feed results into the issue trackers used by dev teams

A better (security) definition of an API

An API consists of 3 parts:

(1) Hostname
e.g. example.com, uat.bigcorp.com

(2) Path
e.g. /api/v2/users/all , /v1/cart/addItem

(3) Method
e.g. POST, PUT, GET, PATCH, DELETE, ...

GET to `example.com/v2/users/all` != DELETE to `example.com/v2/users/all`

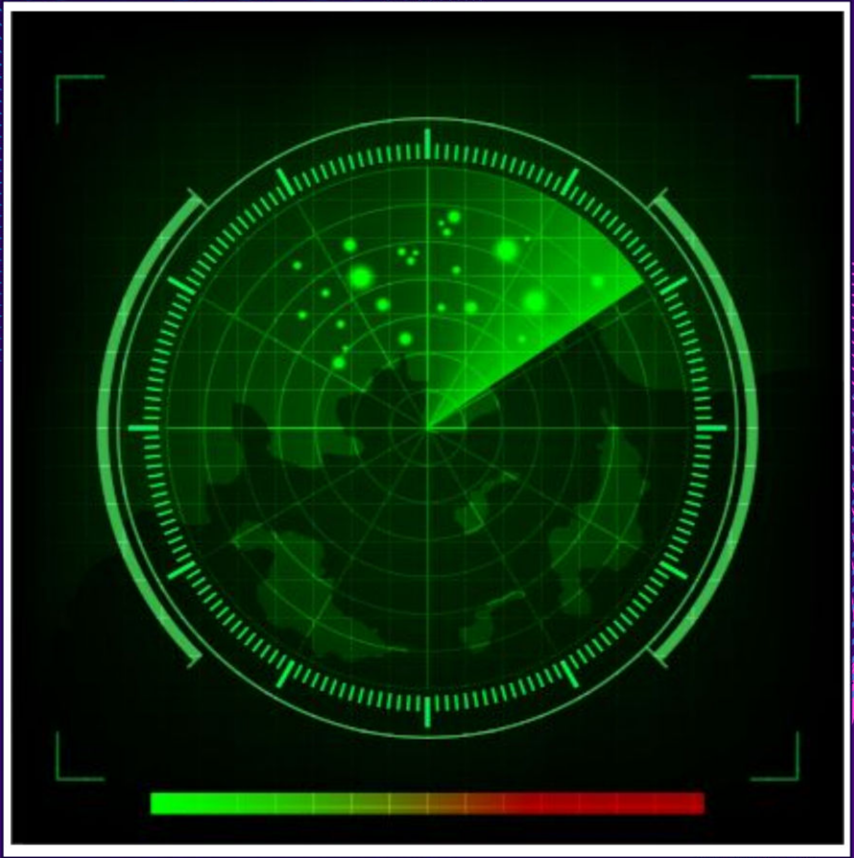
POST to `uat.example.com/v2/user/admin` != POST to `example.com/v2/user/admin`

03

Attacking APIs

Recon

Finding APIs to attack



Passive Recon

Gather all the public information you can on potential targets

Attacker

- No interaction with the target
- OSINT / Public information sources
- Google Dorks
  - intitle: inurl: ext: site: filetype:
- DNS / OWASP Amass
- Shodan
  - Search engine of connected devices
- Search for APIs
  - www.programmableweb.com
  - apis.guru
- Github issues/PRs (if FLOSS)
- Stack Overflow posts

Defender

- Not much to do here - it's public info
- You may ***want*** to advertise your API
- "Getting started" pages
- curl examples, Postman collections
- API docs behind a customer login
- Support docs can help attackers too
  - Username format
  - Password complexity
  - Auth method (bearer token, ...)
- **Posture** & **Runtime** & **Testing** aren't in play since no traffic hits your infra

Active Recon

Gather all the public information you can from a targets (play nice)

Attacker

- Interaction with the target is desired
- Initially traffic looks harmless or clumsy
- Start with basic nmap scans of target(s)
  - Listening ports esp http/https
- Other clues to APIs
  - robots.txt - disallowed URLs
  - DevTools - network tab / XHR / Memory / Performance
- Local proxy (Burp/Zap) for API backed websites / mobile apps
- Bruteforce URLs (dirbuster, dirb, Gobuster)
- Kiterunner - API focused bruteforce

Defender

- Pretty hard to filter from Internet background radiation (noise)
- For SPAs, DevTools are just a fact of life
- Review items pointing to your API like robots.txt
- Nmap scans are detectable but VERY common
- Bruteforce activity stands out if real time monitoring is sufficient
- Kiterunner should trip API monitoring if in place
- **Posture** - focus efforts
  Runtime - discover active recon
  Testing - proactive, not much for Recon

Discovery

Understanding your API target

Discovery

You have target(s), now how to use them legitimately

Attacker

- Learn how to make legitimate requests
  - Especially how to authenticate
- Look for
  - API documentation
  - "Getting Started" guides
  - What the API does / why created
- Spec files (Swagger, OpenAPI, RAML, Postman collections, WADL, WSDL, ...)
- Clients (upstream proxy them)
- Manually creating a list / Postman collection based on:
  - Bruteforced URLs
  - SPA proxied traffic
  - Kiterunner

Defender

- Traffic mostly looks like someone learning your API
- For SPAs & Mobile
  - Discovery may stand out
  - Your clients already know how to make API calls
- For undocumented APIs, there should be many failed requests
- **Posture** - focus efforts, define internal-only APIs

  Runtime - detect Discovery in certain circumstances

  Testing - proactive, not much for Discovery

Discovery seems easy but can be a time sink

Active Attacks

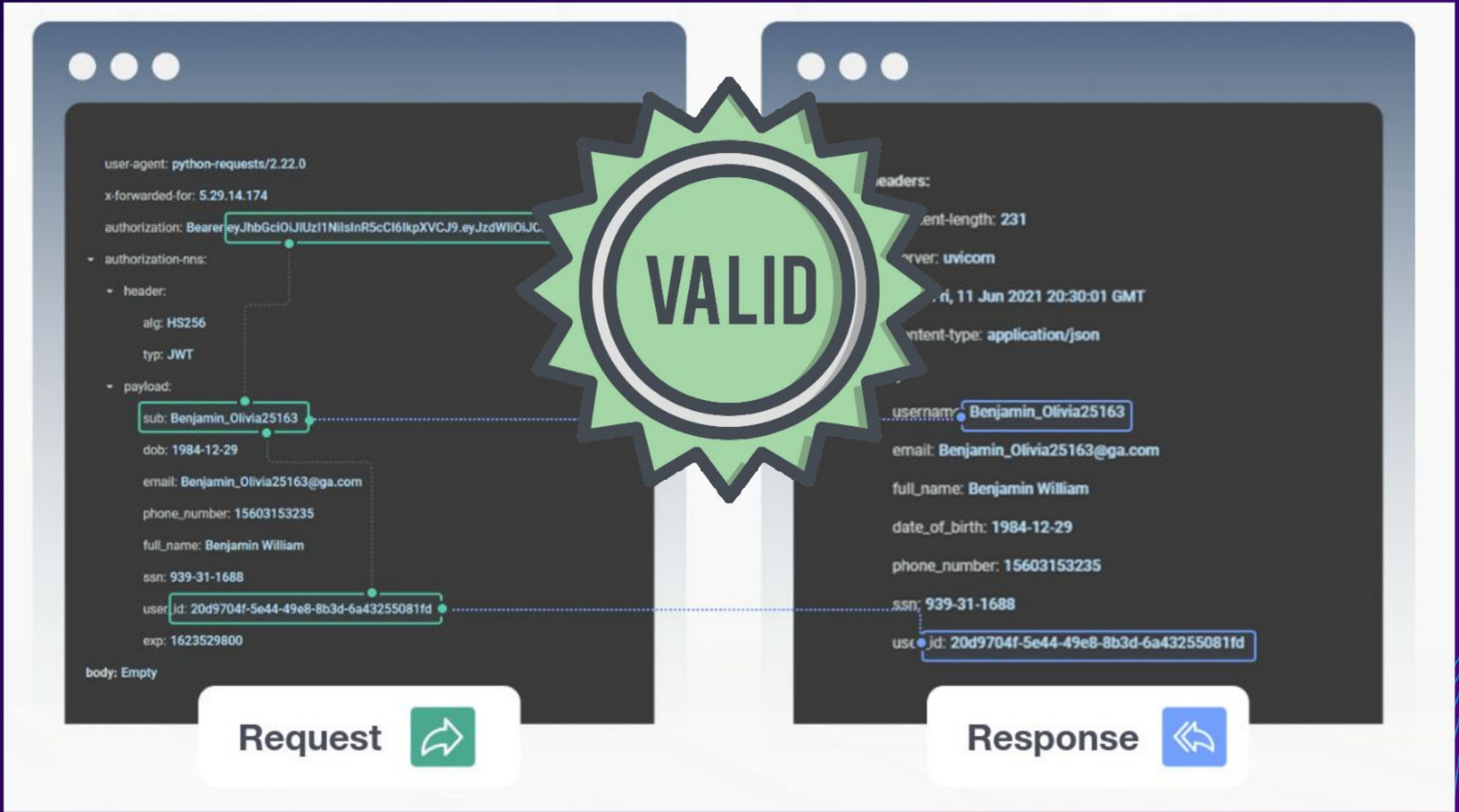
Getting malicious
with your API target

Attacks are grouped into the API Top 10

API-1	Broken Object Level Authorization (BOLA)
API-2	Broken User Authentication
API-3	Excessive Data Exposure
API-4	Lack of Resource & Rate Limiting
API-5	Broken Function Level Authorization

API-6	Mass Assignment
API-7	Security Misconfiguration
API-8	Injection
API-9	Improper Assets Management
API-10	Insufficient Logging & Monitoring

Broken Object Level Authorization (BOLA)

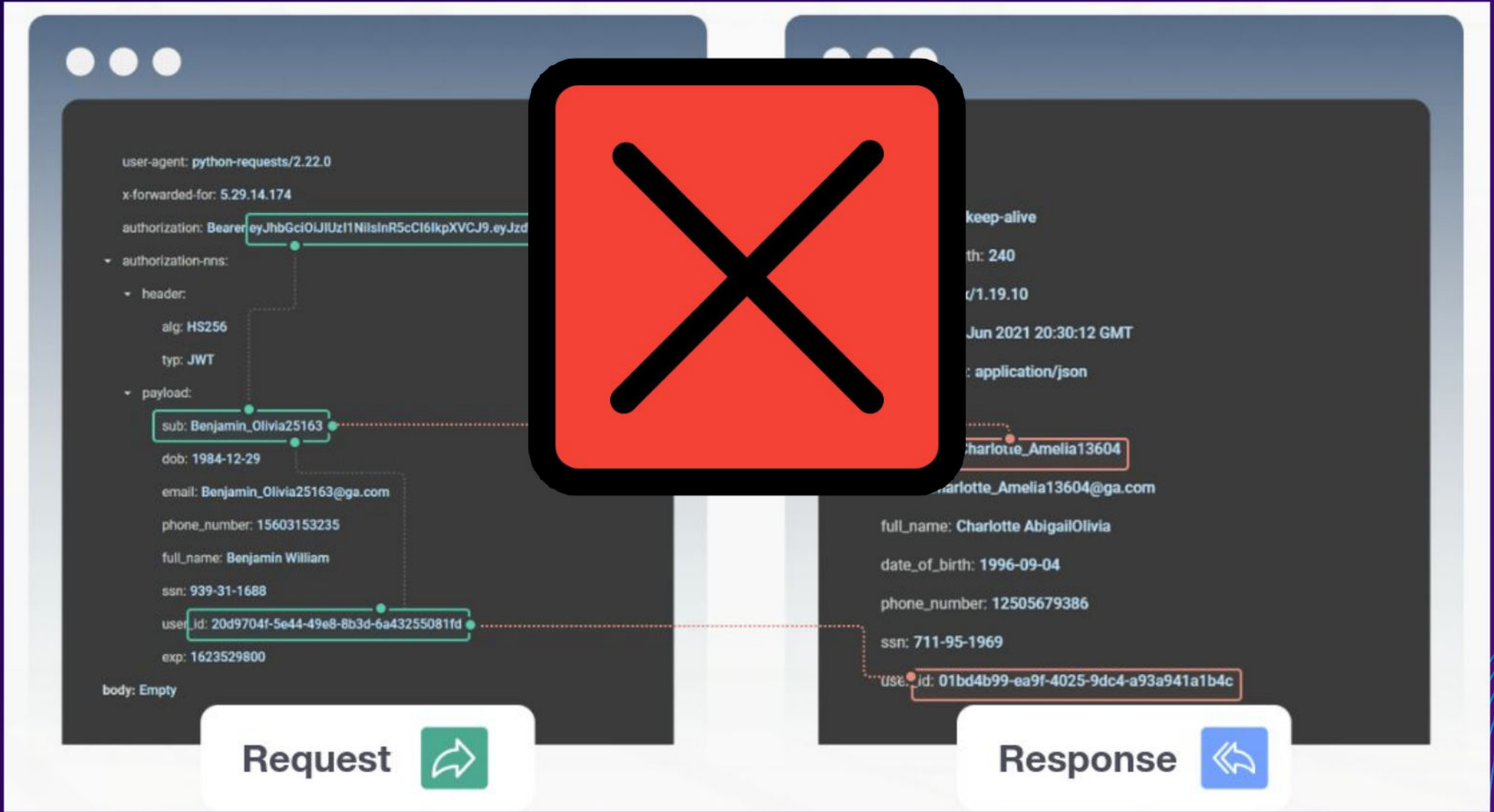


```
user-agent: python-requests/2.22.0
x-forwarded-for: 5.29.14.174
authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJC
▾ authorization-nns:
  ▾ header:
      alg: HS256
      typ: JWT
  ▾ payload:
      sub: Benjamin_Olivia25163
      dob: 1984-12-29
      email: Benjamin_Olivia25163@ga.com
      phone_number: 15603153235
      full_name: Benjamin William
      ssn: 939-31-1688
      user_id: 20d9704f-5e44-49e8-8b3d-6a43255081fd
      exp: 1623529800
body: Empty
```

```
eaders:
ent-length: 231
rver: uvicorn
ri, 11 Jun 2021 20:30:01 GMT
ntent-type: application/json

username: Benjamin_Olivia25163
email: Benjamin_Olivia25163@ga.com
full_name: Benjamin William
date_of_birth: 1984-12-29
phone_number: 15603153235
ssn: 939-31-1688
user_id: 20d9704f-5e44-49e8-8b3d-6a43255081fd
```

Broken Object Level Authorization (BOLA)



```
user-agent: python-requests/2.22.0
x-forwarded-for: 5.29.14.174
authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzd
▾ authorization-nns:
  ▾ header:
      alg: HS256
      typ: JWT
  ▾ payload:
      sub: Benjamin_Olivia25163
      dob: 1984-12-29
      email: Benjamin_Olivia25163@ga.com
      phone_number: 15603153235
      full_name: Benjamin William
      ssn: 939-31-1688
      user_id: 20d9704f-5e44-49e8-8b3d-6a43255081fd
      exp: 1623529800
body: Empty
```

```
keep-alive
th: 240
/1.19.10
Jun 2021 20:30:12 GMT
: application/json

harlotte_Amelia13604
harlotte_Amelia13604@ga.com
full_name: Charlotte AbigailOlivia
date_of_birth: 1996-09-04
phone_number: 12505679386
ssn: 711-95-1969
user_id: 01bd4b99-ea9f-4025-9dc4-a93a941a1b4c
```

Broken Object Level Authorization

One user can access another user’s data or take actions for them

Attacker

- Look at how API resources are structured
  - Change IDs within API calls
  - Can be names (non-numeric)
- Make calls to other IDs/resources with your Auth-N method / token
- Create something as user 1
  - Try to access it as user 2
- Response differences
  - HTTP Response code (404 vs 405)
  - Time to respond
  - Length of response (rare)

Defender

- Detection requires fairly deep inspection of the API calls
  - WAFs will generally fail
  - Shaped like legit request with IDs swapped
- Looking for BOLA can cause increased Auth-Z errors
- 2 similar requests from the same client with different IDs can be found by ML
- **Posture** - focus on most risky APIs
  Runtime - detect BOLA attacks
  Testing - Find BOLA early / pre-prod

Broken User Authentication

Using poor practices in authentication to attack APIs

Attacker

- Bruteforce credentials
- No anti-automation on password resets or MFA/CAPTCHA
- Password Spraying
- Base-64 “protections”
- Low entropy tokens
- JWT weaknesses
  - Captured JWTs
  - None algorithm, no signature
  - Key mismatch, blank password, …
  - Cracking JWT secrets
  - jwt_tool

Defender

- Bruteforce attacks are noisy
- Password spraying is very noisy
- Ensure crypto is used correctly and carefully
- JWT Best Practices RFC
- Consider removing Auth-N from the API
  - Only get tokens through web app
- **Posture** - identify Auth-N APIs
  Runtime -detect brute force, spraying, JWT manipulation
  Testing - identify poor practices early

Excessive Data Exposure

Sometimes developer productivity helps attackers too!

Attacker

- Look for API responses that provide ‘extra’ information
  - Mobile app APIs tend to trust client to filter data
- Look for ‘interesting’ responses
  - Profile pages
  - Linked users
  - Internal meta-data
- Is the data expected part of a larger data object or DB row?
- Can be time consuming to check all possible responses for excessive data

Defender

- Single requests can’t be distinguished from normal traffic
- SAST can help here to avoid “to_json” or similar
- Don’t rely on clients filtering data
- Separate data objects for app and API
- **Posture** - Shows sensitive data, large responses

  Runtime - Detect multi-request data scraping

  Testing - Find verbose responses early

Lack of Resource & Rate Limiting

Failure to provide limits is a recipe for DOS or worse

Attacker

- Add thousands of items, ask for a list
  - Lack of pagination
- Denial of API use (client)
- Fuzzing and bruteforce attacks can discover these
- Modify requests, different client, different IP to bypass limits
- CPU / Memory intensive requests
  - robots.txt or documentation
- Other games to play
  - Switch cAsE
  - Null and other terminators
  - Encoding data
- Too high to make a difference

Defender

- Some requests will look normal but with large responses
- Unusual requests
  - Headers, encoding, terminators, …
- Observability can show usage spikes
- Many bypass methods stand out from normal traffic
- **Posture** - Determine APIs needing limits
  Runtime - Detect anomalous traffic and respond
  Testing - Fuzzing request data can find some issues early

Broken Function Level Authorization

Failure to restrict access by group or role leading to compromise

Attacker

- Focus on APIs with multiple roles/groups
  - Potential for expose backplane
  - Most things have an 'admin'
- Try undocumented HTTP methods
  - PATCH, PUT, POST, DELETE (!)
- Create items with one group/role
  - Interact with those items as a different role
- Bruteforce / guess potential backplane operations
- Experiment with headers, request data to access admin functions

Defender

- Affects APIs with 2+ roles, groups, privilege levels
- Calls to unsupported methods that fail
- Same client, different roles within a short period of time
- Failures for backplane/admin paths
- Unusual requests - headers, body
- **Posture** - Determine APIs with groups, roles, privilege levels

  Runtime - Detect unusual, failing requests or changes in role from a client

  Testing - Conduct Auth-Z testing early

Mass Assignment

Why not accept more data, what could go wrong?

Attacker

- Look for requests that appear to be partial data
  - Make guesses at unsent items
- Look at request/response difference between roles/groups/privilege levels
- Guess / bruteforce multiple values at once (hail mary)
- Error messages or required field messages can provide clues
- Fuzzing can also find issues
- Combine with Broken Function Level Auth-Z to change data for other users
  - Change email/contact details

Defender

- Requests stand out from normal requests with deep inspection
- Large number of failed/invalid requests
- Increased request size
- Increased severity for APIs with different roles/groups/privileges
- **Posture** - Focus on APIs with multi-roles or sensitive data

  Runtime - Requests with extra data, multiple failed/invalid requests

  Testing - Add additional, valid fields to discover early

Security Misconfiguration

A little misconfiguration can go a long way

Attacker

- Check the basics
  - TLS config
  - Info leaks via headers, etc
  - Default credentials, EICAR
  - Use Recon and Discovery
- Verbose errors
  - Purposefully make bad requests
- Misconfigured framework settings
  - Debug mode
- Intermediate devices
  - Determine if WAF, API Gateway, etc is in line
- Call ‘internal’ functions with origin headers e.g. X-Remote-Addr

Defender

- Basic network vuln scanners can find the basics
- Passive traffic monitoring can show header issues, API gateway bypass, many others
- Client with many erroring or malformed requests
- **Posture** - Show weak configuration e.g. API gateway bypass

  Runtime - Unexpected client traffic, multiple errors, malformed or anomalous requests

  Testing - Good for the basics, better if fuzzing is included in tests

Injection

Treat data like code and bad things happen

Attacker

- Place injection strings into
  - Tokens / API keys
  - Headers (esp API specific ones)
  - Query data
  - Data in request body
- Recon/Discovery can help focus what types of injection to try
  - Error messages can also help
- Many good online resources for injections
  - Fuzzing lists
  - OWASP Testing Guide
- 2nd order injections

Defender

- Input validation AND output encoding
- Many failed or malformed requests
- Large number of errors or validation failures at API
- Overly trusting of East/West API calls
- **Posture** - Focus on APIs with sensitive data, East/West APIs

  Runtime - Surge in errors, failed, invalid or malformed requests, control characters in requests

  Testing - Attempt injections early in dev cycle

Improper Assets Management

Know what you have if you want to protect it adequately

Attacker

- You find many misconfiguration issues
- Internal APIs are publicly accessible
- API documentation is inaccurate
- “Hidden”/undocumented APIs
  - Dev/New APIs in production
- Legacy APIs are not decommissioned
- API v minus 1 or more available

Basically,
your pen test was productive and easy

Defender

- Need to know all APIs (host, path, method)
- Classify all data received and sent by APIs
- API Gateway enforced, East/West traffic
- Public vs internal APIs
- **Posture** - Solved with solid posture management

  Runtime - Updates posture as environment changes

  Testing - Not particularly useful here

Insufficient Logging & Monitoring

Change guesses to decisions with data

Attacker

- Fuzzing does not cause a reaction / blocking
  - Assumes control is in scope for testing
- Attacks, especially blatant injections go unnoticed
  - Phone numbers never look like: <script>alert(XSS)</script>
- Mostly, external testers / attackers can only infer the level of logging and monitoring

Defender

- No attacks are seen / noticed
- Diagnosing API issues is difficult
- Unplanned downtime or resource consumption
- **Posture** - Determine the appropriate level of logging per API

  Runtime - Monitoring is what this provides, also can retain traffic for analysis aka quasi-logging

  Testing - Validate logging is working (at best)

Bonus Material

Things that didn't fit nicely into the OWASP API Top 10

Fuzzing

When crafted attacks don’t work, throw the kitchen sink at your target

Attacker

- Send requests altering
  - Values to the extreme (large/small)
  - Negative numbers
  - Decimals for integers
  - Letters for numbers and vice versa
  - Control characters
  - Unicode / non-native characters
- Target fuzzing strings if possible
- Look for changes in
  - Response code
  - Response size
  - Timing
  - Error messages

Defender

- **LOTS OF REQUESTS FROM A SINGLE CLIENT OVER A SHORT PERIOD OF TIME**
- Fuzzing is very noisy on the network
- Spikes in CPU, RAM, request traffic, errors, validation failures
- **Posture** - Not much for Fuzzing
  Runtime - Easily detect fuzzing traffic
  Testing - Fuzzing as a normal part of testing to find issues early (pre-prod)

Structural vs Data Attacks

2 fundamental ways to be naughty with APIs

Structural Attacks

- Modifying the structure of a request
  - Repeating data structures
  - Adding non-printing characters e.g. spaces, tabs, null characters between data elements
  - Removing portions of the data structure
- Messing with the structure of the request only - data provided is legit
- QA / HTTP testing tools generally normalize the structure so won't work
- Custom craft HTTP requests (Python requests library) or use a local proxy like Zap or Burpsuite

Data Attacks

- Modifying the data in a request
  - Substituting fuzzing / injection data for legit data values
  - Providing unexpected or overly large / small data values
- Structure of the request is not modified
- What most fuzzing and injections attacks look like - changing data without changing the structure
- QA / HTTP testing tools can be leveraged to automate these attacks

Special Notes on GraphQL

GraphQL is a special beast but many things are the same

Same from GraphQL

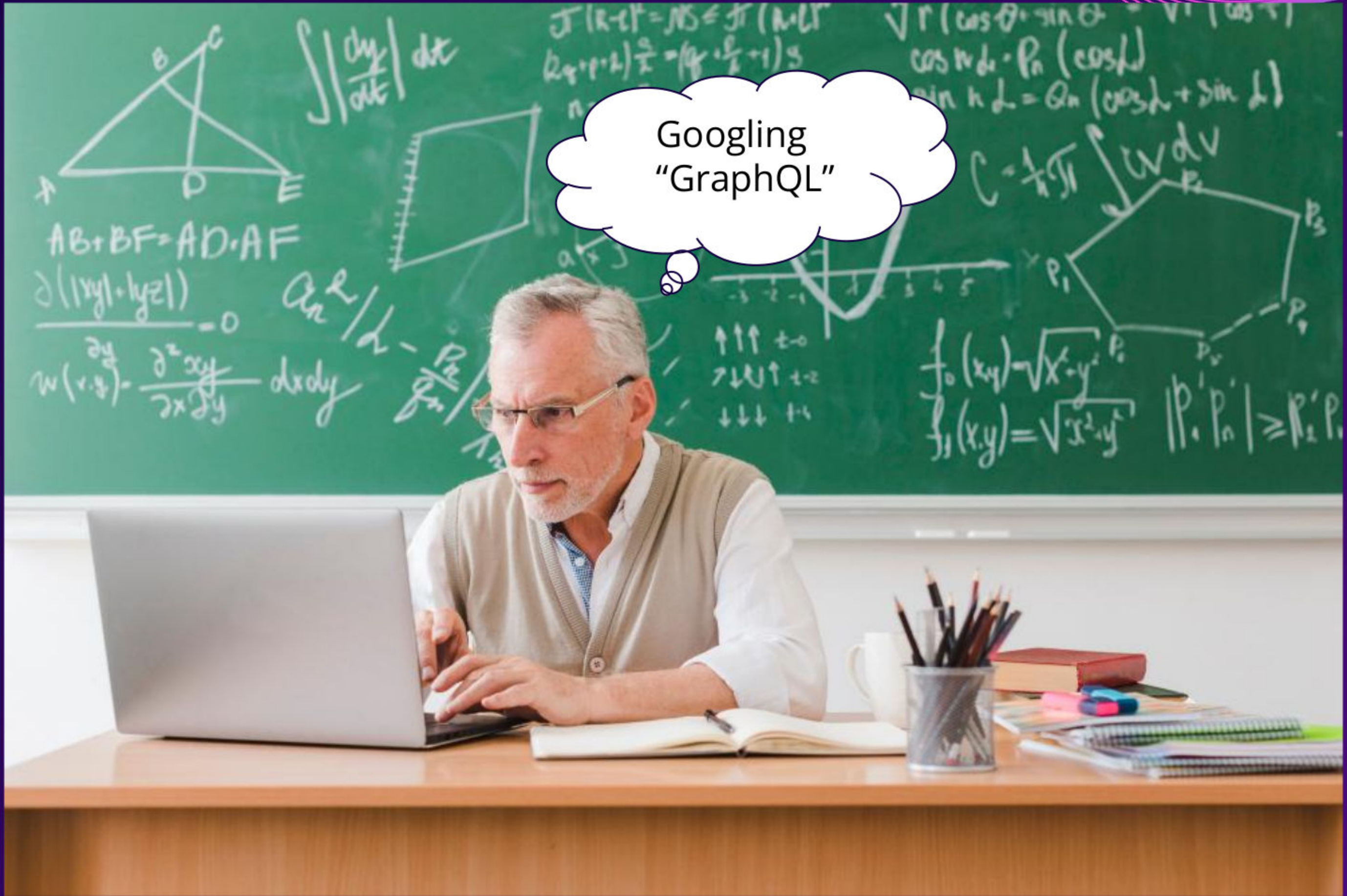
- Recon (Passive / Active)
- Discovery
- Bruteforcing API paths
- Using a local proxy e.g. Burpsuite / Zap
  - Install GraghQL plugins
- Documentation / “Getting Started”

Different for GraphQL

- Introspection to learn the APIs schema
  - Often disabled at the API
- GraphQL is a query language
  - Clients define the data they want
  - Opposite of defined requests & responses of REST APIs
- Gaining popularity as clients aren’t bound to fixed data structures
  - Client can change without need for API changes

https://github.com/dolevf/Damn-Vulnerable-GraphQL-Application

GraphQL - left as an exercise for the student



Googling "GraphQL"

04

Conclusion

Key Takeaways for API testers

(1) **Knowledge of how to test web apps prepares you for most of API testing**

If you need some help, look at the OWASP Testing Guide

(2) **Some special knowledge and tools are needed for parts of API testing**

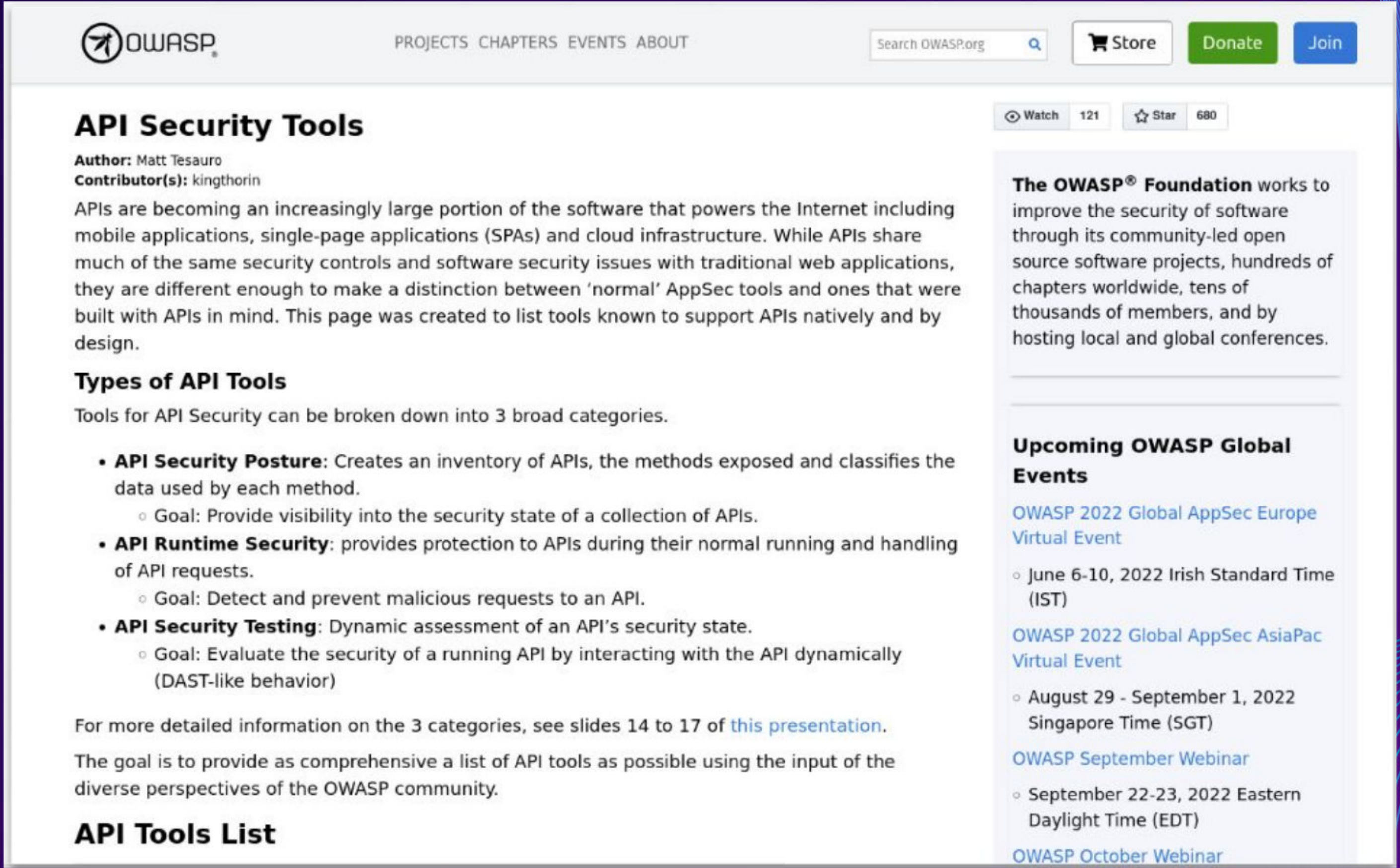
More on this later

(3) **Gaps in AppSec controls coverage and framework shortfalls lead to security shortfalls**

API testing is likely to be “productive”

Key Takeaways for API testers

https://owasp.org/www-community/api_security_tools



API Security Tools

Author: Matt Tesauro
Contributor(s): kingthorin

APIs are becoming an increasingly large portion of the software that powers the Internet including mobile applications, single-page applications (SPAs) and cloud infrastructure. While APIs share much of the same security controls and software security issues with traditional web applications, they are different enough to make a distinction between 'normal' AppSec tools and ones that were built with APIs in mind. This page was created to list tools known to support APIs natively and by design.

Types of API Tools

Tools for API Security can be broken down into 3 broad categories.

- **API Security Posture**: Creates an inventory of APIs, the methods exposed and classifies the data used by each method.
  - Goal: Provide visibility into the security state of a collection of APIs.
- **API Runtime Security**: provides protection to APIs during their normal running and handling of API requests.
  - Goal: Detect and prevent malicious requests to an API.
- **API Security Testing**: Dynamic assessment of an API's security state.
  - Goal: Evaluate the security of a running API by interacting with the API dynamically (DAST-like behavior)

For more detailed information on the 3 categories, see slides 14 to 17 of this presentation.

The goal is to provide as comprehensive a list of API tools as possible using the input of the diverse perspectives of the OWASP community.

API Tools List

The OWASP® Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Upcoming OWASP Global Events

OWASP 2022 Global AppSec Europe Virtual Event

- June 6-10, 2022 Irish Standard Time (IST)

OWASP 2022 Global AppSec AsiaPac Virtual Event

- August 29 - September 1, 2022 Singapore Time (SGT)

OWASP September Webinar

- September 22-23, 2022 Eastern Daylight Time (EDT)

OWASP October Webinar

Key Takeaways for API defenders

The existing AppSec program and controls have API Security gaps to fill

Risk	Posture	Runtime	Testing
Broken Object Level Authorization	💪 weak	💪	💪
Broken User Authentication	💪 weak	💪	💪
Excessive Data Exposure	💪	💪	💪
Lack of Resource & Rate Limiting	💪	💪	💪
Broken Function Level Authorization	💪 weak	💪	💪

Key Takeaways for API defenders

The existing AppSec program and controls have API Security gaps to fill

Risk	Posture	Runtime	Testing
Mass Assignment	🚫	💪	💪
Security Misconfiguration	💪	💪	💪
Injection	💪 weak	💪	💪
Improper Assets Management	💪⬆	💪	🚫
Insufficient Logging & Monitoring	💪	💪⬆	🚫

Sorry about the firehose

THANKS!

Do you have any questions?

matt.tesauro@owasp.org

Deck will be posted at:
https://www.slideshare.net/mtesauro