API's Penetration Testing

A play book for API's PenTest

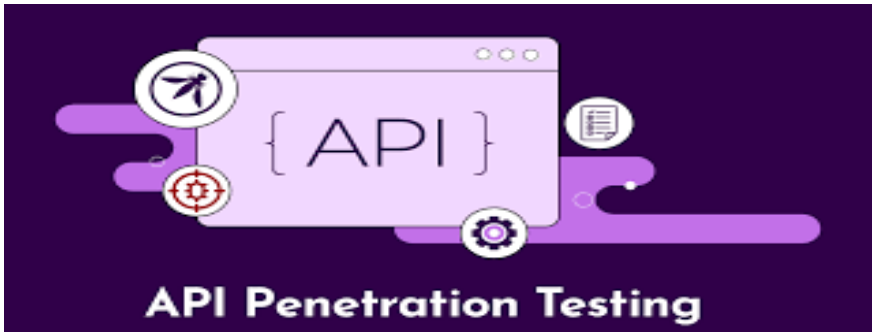


Table of Contents

Passive Reconnaissance :

-Google Hacking:

You can use google to find information about your target ,for Example:

Google Query Parameters:

Query operator	Purpose
Intitle	Searches page titles
Inurl	Searches for words in the URL
Filetype	Searches for desired file types
Site	Limits a search to specific sites

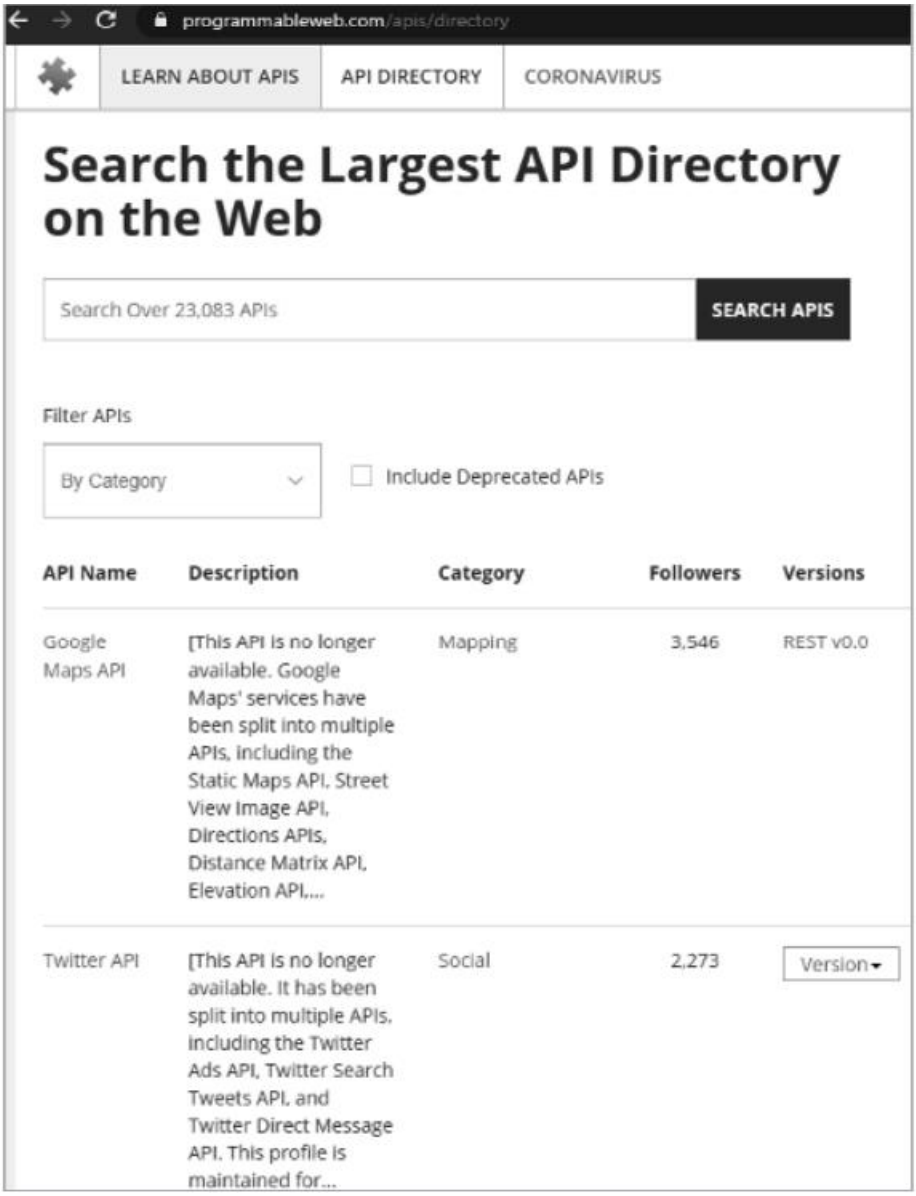
-Offensive Security's Google Hacking Database:

https://www.exploit-db.com/google-hacking-database
some useful API queries from the GHDB:

Google hacking query	Expected results
inurl:"/wp-json/wp/v2/users"	Finds all publicly available WordPress API user directories.
intitle:"index.of" intext:"api.txt"	Finds publicly available API key files.
inurl:"/includes/api/" intext:"index of /"	Finds potentially interesting API directories.
ext:php inurl:"api.php?action="	Finds all sites with a XenAPI SQL injection vulnerability. (This query was posted in 2016; four years later, there were 141,000 results.)
intitle:"index of" api_key OR "api key" OR apiKey	poolLists potentially exposed API keys.

-ProgrammableWeb's API Search Directory:

ProgrammableWeb (https://www.programmableweb.com) is the go-to source for API-related information. To learn about APIs, you can use its API University. To gather information about your target.



-Shodan:

Like with Google dorks, you can search Shodan casually by entering your target's domain name or IP addresses; alternatively, you can use search parameters as you would when writing Google queries. Table 6-3 shows some useful Shodan queries.

Shodan Query Parameters:

Shodan queries	Purpose

hostname:"targetname.com"	Using hostname will perform a basic Shodan search for your target's domain name. This should be combined with the following queries to get results specific to your target.
"content-type: application/json"	APIs should have their content-type set to JSON or XML. This query will filter results that respond with JSON.
"content-type: application/xml"	This query will filter results that respond with XML.
"200 OK"	You can add "200 OK" to your search queries to get results that have had successful requests. However, if an API does not accept the format of Shodan's request, it will likely issue a 300 or 400 response.
"wp-json"	This will search for web applications using the WordPress API.

-OWASP Amass:

Amass is a command line tool that can map a target's external network by collecting OSINT from over 55 different sources. You can set it to perform passive or active scans.

The following is a passive scan of twitter.com, with grep used to show only API-related results:

```
$ amass enum -passive -d twitter.com |grep api
```

```
legacy-api.twitter.com
api1-backup.twitter.com
```

api3-backup.twitter.com
tdapi.twitter.com
failover-urls.api.twitter.com
cdn.api.twitter.com
pulseone-api.smfc.twitter.com
urls.api.twitter.com
api2.twitter.com
apistatus.twitter.com
apiwiki.twtter.com

The user's guide can be found here:
https://github.com/OWASP/Amass/blob/master/doc/user_guide.md

An example configuration file can be found here:
https://github.com/OWASP/Amass/blob/master/examples/config.ini

The Amass tutorial can be found here:
https://github.com/OWASP/Amass/blob/master/doc/tutorial.md

-Exposed Information on GitHub:

Regardless of whether your target performs its own development, it's worth checking GitHub (https://github.com) for sensitive information disclosure. Developers use GitHub to collaborate on software projects. Searching GitHub for OSINT could reveal your target's API capabilities, documentation, and secrets, such as admin-level API keys, passwords, and tokens, which could be useful during an attack. Begin by searching GitHub for your target organization's name paired with potentially sensitive types of information, such as "api-key," "password," or "token." Then investigate the various GitHub repository tabs to discover API endpoints and potential weaknesses. Analyze the source code in the Code tab, find software bugs in the Issues tab, and review proposed changes in the Pull requests tab.

Code:

Code contains the current source code, README files, and other files . This tab will provide you with the name of the last developer who committed to the given file, when that commit happened, contributors, and the actual source code.

Issues:

The Issues tab is a space where developers can track bugs, tasks, and feature requests. If an issue is open, there is a good chance that the vulnerability is still live within the code.

Pull Requests:

The Pull requests tab is a place that allows developers to collaborate on changes to the code. If you review these proposed changes, you might sometimes get lucky and find an API exposure that is in the process of being resolved. For example, the developer has performed a pull request to remove an exposed API key from the source code.

Active Recon:

One shortcoming of performing passive reconnaissance is that you're collecting information from secondhand sources. As an API hacker, the best way to validate this information is to obtain information directly from a target by port or vulnerability scanning, pinging, sending HTTP requests, making API calls, and other forms of interaction with a target's environment. This section will focus on discovering an organization's APIs using detection scanning, hands-on

analysis, and targeted scanning. The lab at the end of the chapter will show these techniques in action.

The Active Recon Process:

The active recon process should lead to an efficient yet thorough investigation of the target and reveal any weaknesses you can use to access the system.

Phase One: Detection Scanning:

The goal of detection scanning is to reveal potential starting points for your investigation. Begin with general scans meant to detect hosts, open ports, services running, and operating systems currently in use, as described in the "Baseline Scanning with Nmap" section of this chapter. APIs use HTTP or HTTPS, so as soon as your scan detects these services, let the scan contenue to run and move into phase two.

Phase Two: Hands-on Analysis:

Hands-on analysis is the act of exploring the web application using a browser and API client. Aim to learn about all the potential levers you can interact with and test them out. Practically speaking, you'll examine the web page, intercept requests, look for API links and documentation, and develop an understanding of the business logic involved. You should usually consider the application from three perspectives: guests, authenticated users, and site administrators. Guests are anonymous users likely visiting a site for the first time. If the site hosts public Information and does not need to authenticate users, it may only have guest users. Authenticated users have gone through some registration process and have been granted a certain level of access. Administrators have the privileges to manage and maintain the API. Your first step is to visit the website in a browser, explore the site, and consider it from these perspectives.

Here are some considerations for each user group:

Guest:

How would a new user use this site? Can new users interact with the API? Is API documentation public? What actions can this group perform?

Authenticated User:

What can you do when authenticated that you couldn't do as a guest? Can you upload files? Can you explore new sectons of the web application? Can you use the API? How does the web application recognize that a user is authenticated?

Administrator:

Where would site administrators log in to manage the web app? What is in the page source? What comments have been left around various pages? What programming languages are in use? What sections of the website are under development or experimental?

Next, analyze the app as a hacker by intercepting the HTTP traffic with Burp Suite. When you use the web app's search bar or attempt to authenticate, the app might be using API requests to perform the requested action, and you'll see those requests in Burp Suite.

Finding Hidden Paths in Robots.txt:

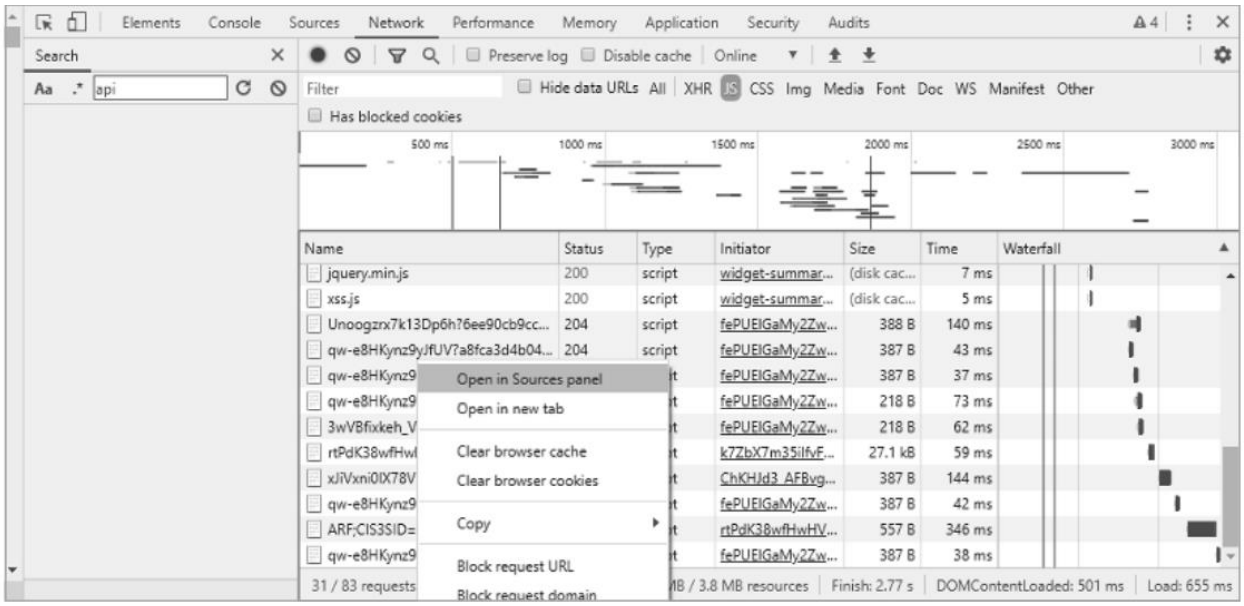
Robots.txt is a common text file that tells web crawlers to omit results from the search engine findings. Ironically, it also serves to tell us which paths the target wants to keep secret. You can find the robots.txt file by navigating to the target's /robots.txt directory (for example, https://www.twitter.com/robots.txt).

The following is an actual robots.txt file from an active web server, complete with a disallowed /api/ path:

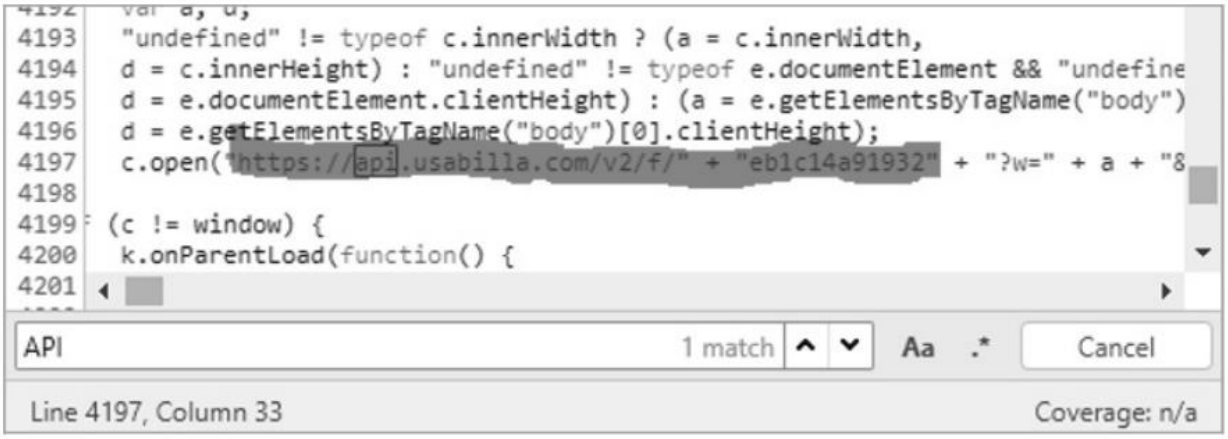
```
User-agent: *
Disallow: /appliance/
Disallow: /login/
Disallow: /api/
Disallow: /files/
```

Finding Sensitive Information with Chrome DevTools:

Begin by opening your target page and then open Chrome DevTools with F12 or CTRL-SHIFT-I. Adjust the Chrome DevTools window until you have enough space to work with. Select the Network tab and then refresh the page. Now look for interesting files (you may even find one titled "API"). Right-click any JavaScript files that interest you and click Open in Sources Panel to view their source code. Alternatively, click XHR to find see the Ajax requests being made:

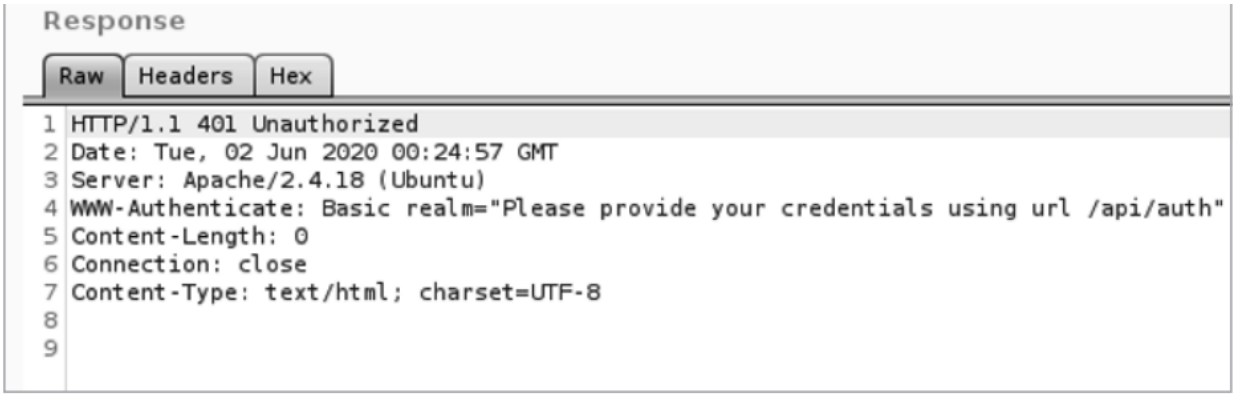


Search for potentially interesting lines of JavaScript. Some key terms to search for include “API,” “APIkey,” “secret,” and “password.” For example, illustrates how you could discover an API that is nearly 4,200 lines deep within a script:



Validating APIs with Burp Suite:

Burp Suite can also be your primary mode of validating your discoveries. To validate APIs using Burp, intercept an HTTP request sent from your browser and then use the Forward button to send it to the server. Next, send the request to the Repeater module, where you can view the raw web server response . As you can see in this example, the server returns a 401 Unauthorized status code, which means that I am not authorized to use the API. Compare this request to one that is for a nonexistent resource, and you will see that your target typically responds to nonexistent resources in a certain way. (To request a nonexistent resource, simply add various gibberish to the URL Discovery 143 path in Repeater, like GET /user/test098765. Send the request in Repeater and see how the web server responds. Typically, you should get a 404 or similar response.):



Brute-Forcing URIs with Gobuster:

following example uses an API-specific wordlist to find the directories on an IP address:
$ gobuster dir -u http://192.168.195.132:8000 -w
/home/hapihacker/api/wordlists/common_apis_160

```
 Gobuster by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.195.132:8000
 [+] Method:                  GET
 [+] Threads:                 10
 [+] Wordlist:                /home/hapihacker/api/wordlists/common_apis_160
 [+] Negative Status codes:   404
 [+] User Agent:              gobuster
 [+] Timeout:                 10s
===============================================================
 09:40:11 Starting gobuster in directory enumeration mode
===============================================================
 /api (Status: 200) [Size: 253]
 /admin (Status: 500) [Size: 1179]
 /admins (Status: 500) [Size: 1179]
 /login (Status: 200) [Size: 2833]
 /register (Status: 200) [Size: 2846]
```

Once you find API directories like the /api directory shown in this output, either by crawling or brute force, you can use Burp to investigate them further. Gobuster has additional options, and you can list them using the -h option:

$ gobuster dir -h

If you would like to ignore certain response status codes, use the option -b. If you would like to see additional status codes, use -x. You could enhance a Gobuster search with the following:

$ gobuster dir -u http://targetaddress/ -w /usr/share/wordlists/api_list/common_apis_160 -x 200,202,301 -b 302

Discovering API Content with Kiterunner:

Kiterunner is the best tool available for discovering API endpoints and resources. Now it's time to put this tool to use.

Kiterunner will try POST POST /api/v1/user/create, mimicking a more realistic request. You can perform a quick scan of your target's URL or IP address like this:

```
$ kr scan http://192.168.195.132:8090 -w ~/api/wordlists/data/kiterunner/routes-large.kite
+--------------------+-------------------------------------------------------------------------
------------------+-------------------------------------------------------------------
| SETTING            | VALUE                                                                  |
+--------------------+-------------------------------------------------------------------------
------------------+-------------------------------------------------------------------
| delay              | 0s                                                                     |
| full-scan          | false                                                                  |
| full-scan-requests | 1451872                                                                |
| headers            | [x-forwarded-for:127.0.0.1]                                            |
| kitebuilder-apis   | [/home/hapihacker/api/wordlists/data/kiterunner/routes-large.kite]     |
| max-conn-per-host  | 3                                                                      |
| max-parallel-host  | 50                                                                     |
| max-redirects      | 3                                                                      |
| max-timeout        | 3s                                                                     |
| preflight-routes   | 11                                                                     |
| quarantine-threshold | 10                                                                   |
| quick-scan-requests | 103427                                                                |
| read-body          | false                                                                  |
| read-headers       | false                                                                  |
| scan-depth         | 1                                                                      |
| skip-preflight     | false                                                                  |
| target             | http://192.168.195.132:8090                                            |
| total-routes       | 957191                                                                 |
| user-agent         | Chrome. Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.96 Safari/537.36 |
+--------------------+-------------------------------------------------------------------------

POST    400 [    941,   46,  11] http://192.168.195.132:8090/trade/queryTransationRecords
0cf689f783e6dab12b6940616f005ecfcb3074c4
POST    400 [    941,   46,  11] http://192.168.195.132:8090/event
0cf6890acb41b42f316e86efad29ad69f54408e6
GET     301 [    243,    7,  10] http://192.168.195.132:8090/api-docs -> /api-docs/?group=63578
528&route=33616912 0cf681b5cf6c877f2e620a8668a4abc7ad07e2db
```

As you can see, Kiterunner will provide you with a list of interesting paths. The fact that the server is responding uniquely to requests to certain /api/ paths indicates that the API exists. Note that we conducted this scan without any authorization headers, which the target API likely requires. I will demonstrate how to use Kiterunner with authorization headers in Chapter 7. If

you want to use a text wordlist rather than a .kite file, use the brute option with the text file of your choice:

```
$ kr brute -w ~/api/wordlists/data/automated/nameofwordlist.txt
```

If you have many targets, you can save a list of line-separated targets as a text file and use that file as the target. You can use any of the following line-separated URI formats as input:

Test.com
Test2.com:443
http://test3.com
http://test4.com
http://test5.com:8888/api

One of the coolest Kiterunner features is the ability to replay requests. Thus, not only will you have an interesting result to investigate, you will also be able to dissect exactly why that request is interesting. In order to replay a request, copy the entire line of content into Kiterunner, paste it using the kb replay option, and include the wordlist you used:

```
$ kr kb replay "GET 414 [ 183, 7, 8] http://192.168.50.35:8888/api/privatisations/ count 0cf6841b1e7ac8badc6e237ab300a90ca873d571" -w ~/api/wordlists/data/kiterunner/routeslarge.kite
```

-Endpoint Analysis:

Finding Request Information:

Before you craft requests to an API, you'll need an understanding of its endpoints, request parameters, necessary headers, authentication requirements, and administrative functionality. Documentation will often point us to those elements. Therefore, to succeed as an API hacker, you'll need to know how to read and use API documentation, as well as how to find it. Even better, if you can find a specification for an API, you can import it directly into Postman to automatically craft requests. When you're performing a black box API test and the documentation is truly unavailable, you'll be left to reverse engineer the API requests on your own. You will need to thoroughly fuzz your way through the API to discover endpoints, parameters, and header requirements in order to map out the API and its functionality.

Finding Information in Documentation:

As you know by now, an API's documentation is a set of instructions published by the API provider for the API consumer. Because public and partner APIs are designed with self-service in mind, a public user or a partner should be able to find the documentation, understand how to use the API, and do so without assistance from the provider. It is quite common for the documentation to be located under directories like the following:

https://example.com/docs
https://example.com/api/docs
https://docs.example.com
https://dev.example.com/docs
https://developer.example.com/docs
https://api.example.com/docs
https://example.com/developers/documentation

When the documentation is not publicly available, try creating an account and searching for the documentation while authenticated. If you still cannot find the docs, you can use API wordlists on GitHub that can help you discover API documentation through the use of a fuzzing technique called directory brute force (https://github.com/hAPI-hacker/Hacking-APIs). You can use the subdomains_list and the dir_list to brute-force web application subdomains and domains and potentially find API docs hosted on the site. There is a good chance you'll be able to discover documentation during reconnaissance and web application scanning.

Testing Intended Use:

As you proceed, ask yourself these questions:

- What sorts of actions can I take?
- Can I interact with other user accounts?
- What kinds of resources are available?
- When I create a new resource, how is that resource identified?
- Can I upload a file? Can I edit a file?

Performing Privileged Actions
Analyzing API Responses
Finding Information Disclosures
Finding Security Misconfigurations
Verbose Errors
Poor Transit Encryption
Finding Excessive Data Exposures
Finding Business Logic Flaws

ATTACKING AUTHENTICATION:

Classic Authentication Attacks:

Password Brute-Force Attacks
Password Reset and Multifactor Authentication Brute-Force Attacks
Password Spraying
including Base64 Authentication in Brute-Force Attacks
Forging Tokens
Manual Load Analysis
Brute-Forcing Predictable Tokens

JSON Web Token Abuse:

Recognizing and Analyzing JWTs
None Attack
Algorithm Switch Attack
JWT Crack Attack
To perform a JWT Crack attack using JWT_Tool, use the following command:

```
$ jwt_tool -C -d /wordlist.txt
```

The -C option indicates that you'll be conducting a hash crack attack and the -d option specifies the dictionary or wordlist you'll be using against the hash. In this example, the name of my dictionary is wordlist.txt, but you can specify the directory and name of whatever wordlist you would like to use. JWT_Tool will either return "CORRECT key!" for each value in the dictionary or indicate an unsuccessful attempt with "key not found in dictionary."

Fuzzing:

Choosing Fuzzing Payloads
Detecting Anomalies
Fuzzing Wide and Deep
Fuzzing Wide for Improper Assets Management
Testing Request Methods
Fuzzing "Deeper" to Bypass Input Sanitization
Fuzzing for Directory Traversal

EXPLOITING AUTHORIZATION:

-Testing for BOLA(Broken object Level Authorization)

- Swap out your UserA token for another user's token
- Using UserB's token, make the request for UserA's resources
- Create multiple accounts at each privilege level to which you have access
- Using your accounts, create a resource with UserA's account and attempt to interact with it using UserB's

side-channel BOLA disclosures:

Examples of Side-Channel BOLA Disclosures:

Request	Response
GET /api/user/test987123	404 Not Found HTTP/1.1
GET /api/user/hapihacker	405 Unauthorized HTTP/1.1 { }
GET /api/user/1337	405 Unauthorized HTTP/1.1 { }
GET /api/user/phone/2018675309	405 Unauthorized HTTP/1.1 { }

-Testing for BFLA:

- Create, read, update, or delete resources as UserA
- Swap out your UserA token for UserB's.
- Send GET, PUT, POST, and DELETE requests for UserA's resources using UserB's token
- Check UserA's resources to validate changes have been made by using UserB's token

Mass Assignment:

Finding Mass Assignment:
Account Registration
Unauthorized Access
Finding Mass Assignment Variables:
Finding Variables in Documentation
Fuzzing Unknown Variables
Blind Mass Assignment Attacks
Automating Mass Assignment Attacks

Injection:

Discovering Injection Vulnerabilities:

You should attempt injection attacks against all potential inputs and especially within the following:

- API keys
- Tokens
- Headers
- Query strings in the URL
- Parameters in POST/PUT requests

Cross-Site Scripting (XSS)
Cross-API Scripting (XAS)
SQL Injection
NoSQL Injection
Operating System Command Injection

A P P LY I N G E V A S I V E T E C H N I Q U E S:

Evasive Techniques:

- String Terminators
- Case Switching
- Encoding Payloads

Testing Rate Limits:

APIs often include headers like the following to let you know how many more requests you can make before you violate the limit:

x-rate-limit:

x-rate-limit-remaining:

Path Bypass

Origin Header Spoofing:

Some API providers use headers to enforce rate limiting. These origin request headers tell the web server where a request came from. If the client generates origin headers, we could manipulate them to evade rate limiting. Try including common origin headers in your request like the following:

X-Forwarded-For

X-Forwarded-Host

X-Host X-Originating-IP

X-Remote-IP

X-Client-IP

X-Remote-Addr

Rotating IP Addresses in Burp Suite

API's Checklist:

Passive Reconnaissance	Conduct attack surface discovery
	Check for exposed secrets
Active Reconnaissance	Scan for open ports and services
	Discover API endpoints
	Use the application as intended
	Search for API-related directories
Endpoint Analysis	Find and review API documentation
	Reverse engineer the API
	Use the API as intended
	Analyze responses for information disclosures, excessive data exposures, and business logic flaws.
Authentication Testing	Conduct basic authentication testing
	Attack and manipulate API tokens
Conduct Fuzzing	Fuzz all the things
Authorization Testing	Discover resource identification methods
	Test for BOLA(Broken Object Level Authorization
	Test for BFLA(Broken Function Level Authorization
Mass Assignment Testing	Discover standard parameters used in requests

	Test for mass assignment

References and Recourses:

-Hacking API's Book

-A collection of awesome API Security tools and resources:

https://github.com/arainho/awesome-api-security