





**APISEC**  
UNIVERSITY

[My Courses](#) [Tools & Resources](#) [Scan Your API Now](#) [↗ Search](#)



[My Library](#) [Settings](#) [Logout](#)

	<input type="text" value="Search for something..."/>	
---	--	---

API Penetration Testing

/

Categories

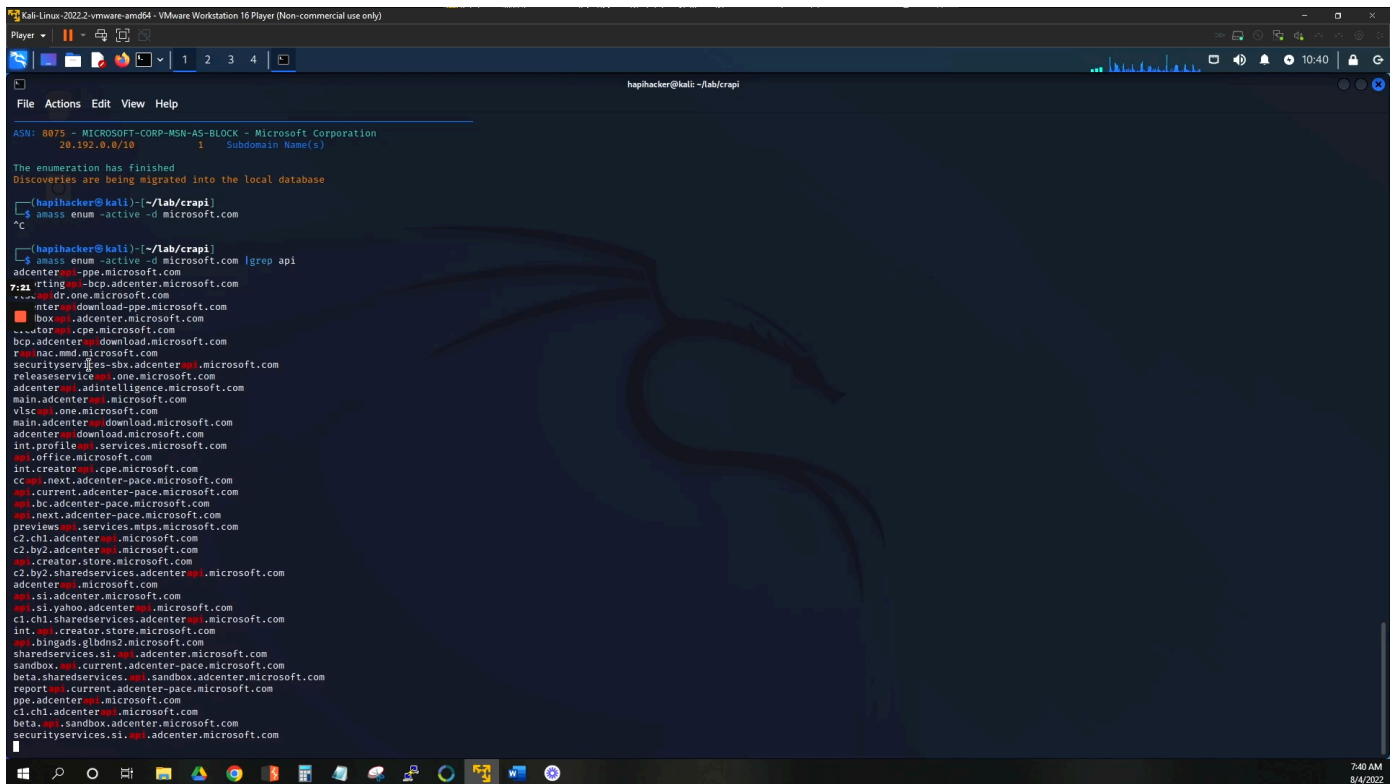
/

API Reconnaissance

/

Active Reconnaissance





## API Reconnaissance

## Lessons

- 1 Introduction to API Reconnaissance
- 2 Passive Reconnaissance
- ▶ Active Reconnaissance
- 4 API Reconnaissance - Quiz



Next Category

◀ Back

Lesson 3 of 4

Next ▶

# Active Reconnaissance

## API Reconnaissance

## Active API Reconnaissance

Active reconnaissance is the process of interacting directly with the target primarily through the use of scanning. We will use our recon to search for our target's APIs and any useful information. During this process you will be scanning systems, enumerating open ports, and finding ports that have services using HTTP. Once you have found systems hosting HTTP, you can open a web browser and investigate the web application. You could find an API being advertised to end users or you may have to dig deeper. Finally, you can scan the web app for API-related directories. Essentially, you will be building out the target's API attack surface. During active recon we will use tools like: nmap, OWASP Amass, gobuster, kiterunner, and DevTools.

### Nmap

Nmap is a powerful tool for scanning ports, searching for vulnerabilities, enumerating services, and discovering live hosts. For API discovery, you should run two Nmap scans in particular: general detection and all port. The Nmap general detection scan uses default scripts (-sC) and service enumeration (-sV) against a target and then saves the output in three formats for later review (-oX for XML, -oN for Nmap, -oG for greppable, or -oA for all three):

```
$ nmap -sC -sV [target address or network range] -oA nameofoutput
```

The Nmap all-port scan will quickly check all 65,535 TCP ports for running services, application versions, and host operating system in use:

```
$ nmap -p- [target address] -oA allportscan
```

As soon as the general detection scan begins returning results, kick off the all-port scan. Then begin your hands-on analysis of the results. You'll most likely discover APIs by looking at the results related to HTTP traffic and other indications of web servers. Typically, you'll find these running on ports 80 and 443, but an API can be hosted on all sorts of different ports. Once you discover a web server, you can perform HTTP enumeration using a Nmap NSE script (use -p to specify which ports you'd like to test).

```
$ nmap -sV --script=http-enum <target> -p 80,443,8000,8080
```

## OWASP Amass

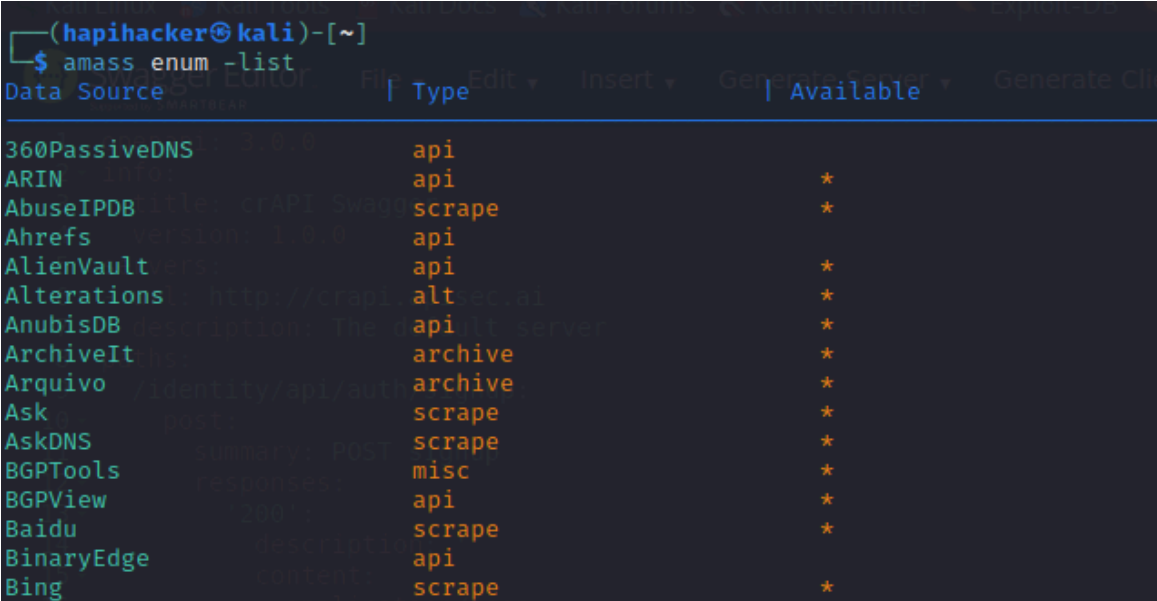
OWASP Amass is a command-line tool that can map a target's external network by collecting OSINT from over 55 different sources. You can set it to perform passive or active scans. If you choose the active option, Amass will collect information directly from the target by requesting its certificate information. Otherwise, it collects data from search engines (such as Google, Bing, and HackerOne), SSL certificate sources (such as GoogleCT, Censys, and FacebookCT), search APIs (such as Shodan, AlienVault, Cloudflare, and GitHub), and the web archive Wayback.

## Making the most of Amass with API Keys

Before diving into using Amass, we should make the most of it by adding API keys to it. Let's obtain a few free API keys to enhance our Amass scans.

First, we can see which data sources are available for Amass (paid and free) by running:

```
$ amass enum -list
```



Data Source	Type	Available
360PassiveDNS	api	
ARIN	api	*
AbuseIPDB	scrape	*
Ahrefs	api	
AlienVault	api	*
Alterations	alt	*
AnubisDB	api	*
ArchiveIt	archive	*
Arquivo	archive	*
Ask	scrape	*
AskDNS	scrape	*
BGPTools	misc	*
BGPView	api	*
Baidu	scrape	*
BinaryEdge	api	
Bing	scrape	*

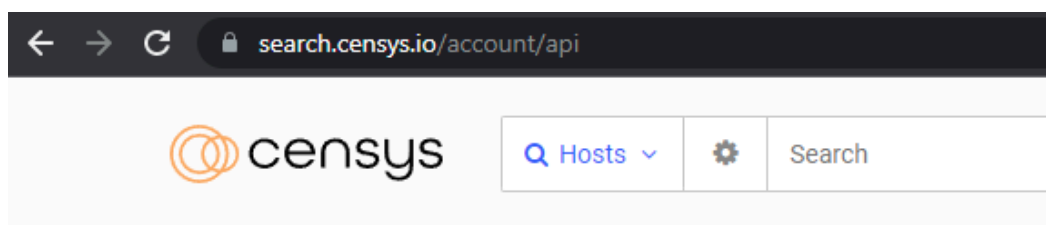
Next, we will need to create a config file to add our API keys to.

```
$ sudo curl
```

```
https://raw.githubusercontent.com/OWASP/Amass/master/examples/config.ini
```

```
> ~/.config/amass/config.ini
```

Now we can update the config.ini. I will demonstrate the process for adding API keys with Censys. Simply visit <https://censys.io/register> and register for a free account. Make sure to use a valid email because you will have to verify for access to your free account.



## My Account

Details API

### API Credentials

Below are the credentials that can be used for accessing [authenticated APIs](#):

API ID	a7206336-8a1a-4387-8fe8-29a4dbb493f7	
Secret	u9bdkJ9oGcI1GkhPlWqrKRElFbHjRzbj	

RESET MY API SECRET

Once you have obtained your API ID and Secret, edit the config.ini file and add the credentials to the file.

```
$ sudo nano ~/.config/amass/config.ini
```

```
hapihacker@kali: ~/.config/amass
File Actions Edit View Help
GNU nano 6.2 config.ini
# https://censys.io (Paid/Free-trial)
#[data_sources.Censys]
#ttl = 10080
#[data_sources.Censys.Credentials]
#apikey =a7206336-8a1a-4387-8fe8-29a4dbb493f7
#secret =u9bdkJ9oGcI1GkhPlWqrKRElFbHjRzbj
```

Also, as with any credentials make sure not to share them like I just did. If you did share them then simply use the "Reset My API Secret" button back on Censys.io. You can repeat this process with many free accounts and API keys, then you will make OWASP Amass into a powerhouse for API reconnaissance.

```
$ amass enum -active -d target-name.com |grep api
legacy-api.target-name.com
api1-backup.target-name.com
api3-backup.target-name.com
```

This scan could reveal many unique API subdomains, including legacy-api.target-name.com. An API endpoint named legacy could be of particular interest because it seems to indicate an improper asset management vulnerability.

Amass has several useful command-line options. Use the intel command to collect SSL certificates, search reverse Whois records, and find ASN IDs associated with your target. Start by providing the command with target IP addresses

```
$ amass intel -addr [target IP addresses]
```

If this scan is successful, it will provide you with domain names. These domains can then be passed to intel with the whois option to perform a reverse Whois lookup:

```
$ amass intel -d [target domain] -whois
```

This could give you a ton of results. Focus on the interesting results that relate to your target organization. Once you have a list of interesting domains, upgrade to the enum subcommand to begin enumerating subdomains. If you specify the -passive option, Amass will refrain from directly interacting with your target:

```
$ amass enum -passive -d [target domain]
```

The active enum scan will perform much of the same scan as the passive one, but it will add domain name resolution, attempt DNS zone transfers, and grab SSL certificate information:

```
$ amass enum -active -d [target domain]
```

To up your game, add the -brute option to brute-force subdomains, -w to specify the API\_superlist wordlist, and then the -dir option to send the output to the directory of your choice:

```
$ amass enum -active -brute -w /usr/share/wordlists/API_superlist -d [target domain] -dir [directory name]
```

## Directory Brute-force with Gobuster

Gobuster can be used to brute-force URIs and DNS subdomains from the command line. (If you prefer a graphical user interface, check out OWASP's Dirbuster.) In Gobuster, you can use wordlists for common directories and subdomains to automatically request every item in the wordlist and send them to a web server and filter the interesting server responses. The results generated from Gobuster will provide you with the URL path and the HTTP status response codes. (While you can brute-force URIs with Burp Suite's Intruder, Burp Community Edition is much slower than Gobuster.)

Whenever you're using a brute-force tool, you'll have to balance the size of the wordlist and the length of time needed to achieve results. Kali has directory wordlists stored under /usr/share/wordlists/dirbuster that are thorough but will take some time to complete. Instead, you can use an API-related wordlist, which will speed up your Gobuster scans since the wordlist is relatively short and only contains directories related to APIs.

The following example uses an API-specific wordlist to find the directories on an IP address:

```
$ gobuster dir -u target-name.com: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
```

```
://targetaddress/ -w /usr/share/wordlists/api_list/common_apis_160 -x 200,202,301 -b 302
```


Gobuster provides a quick way to enumerate active URLs find API paths.

## Kiterunner

Kiterunner is an excellent tool that was developed and released by Assetnote. Kiterunner is currently the best tool available for discovering API endpoints and resources. While directory brute force tools like Gobuster/Dirbuster/ work to discover URL paths, it typically relies on standard HTTP GET requests. Kiterunner will not only use all HTTP request methods common with APIs (GET, POST, PUT, and DELETE) but also mimic common API path structures. In other words, instead of requesting GET /api/v1/user/create, Kiterunner will try 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://127.0.0.1 -w ~/api/wordlists/data/kiterunner/routes-large.kite
```

```
(hapihacker@kali)-[~/lab/vapi/vapi]
$ sudo kr scan 127.0.0.1 -w /usr/share/wordlists/routes-large.kite
```



```
+-----+
+
| SETTING      | VALUE
+-----+
+
| delay         | 0s
|
| full-scan     | false
|
| full-scan-requests | 2903722
|
| headers       | [x-forwarded-for:127.0.0.1]
|
| kitebuilder-apis | [/usr/share/wordlists/routes-large.kite]
```

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 <target> -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]
```

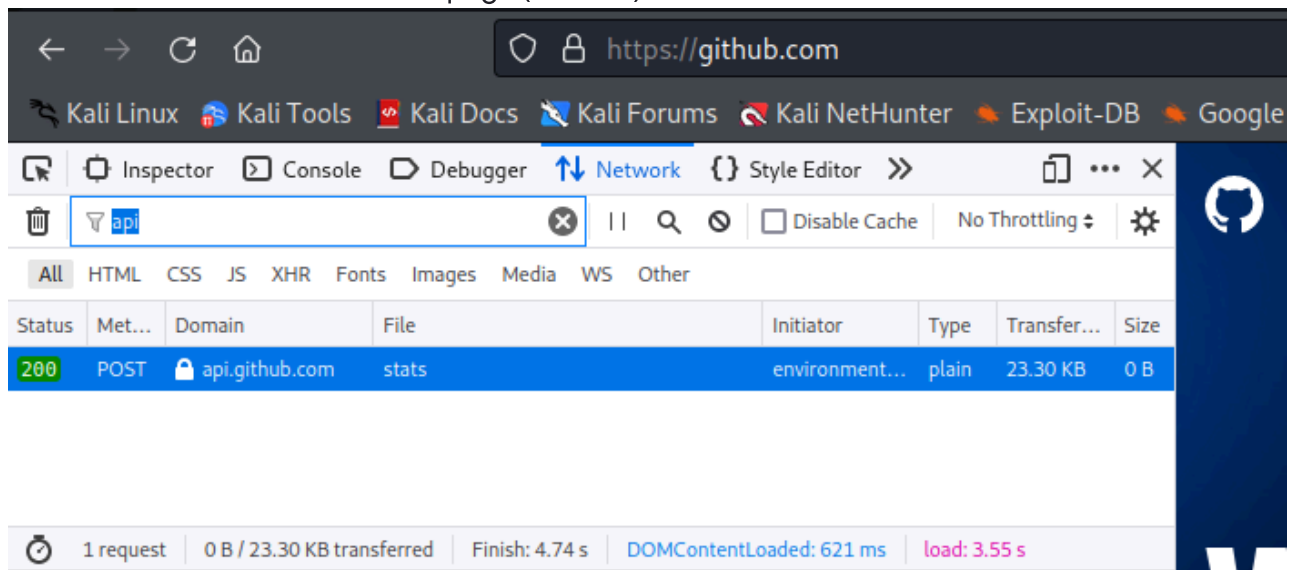


```
://192.168.50.35:8888/api/privatisations/count
Ocf6841b1e7ac8badc6e237ab300a90ca873d571" -w
~/api/wordlists/data/kiterunner/routes-large.kite
```

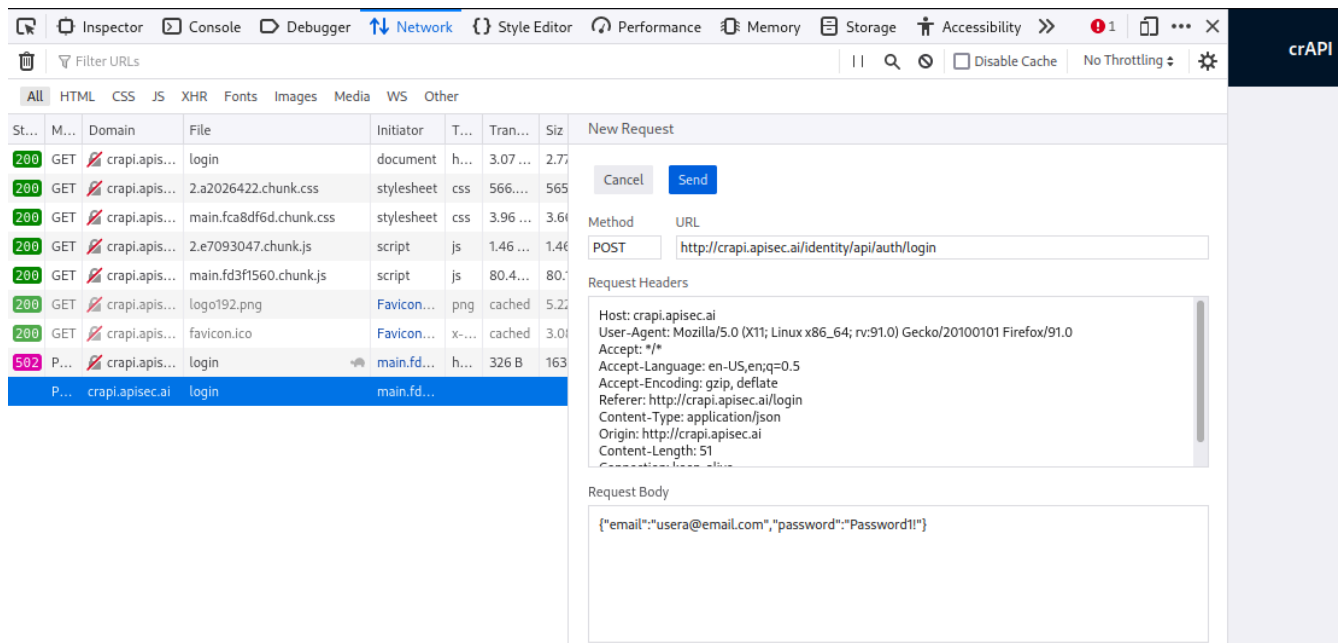
Running this will replay the request and provide you with the HTTP response. You can then review the contents to see if there is anything worthy of investigation. I normally review interesting results and then pivot to testing them using Postman and Burp Suite.

## DevTools

DevTools contains some highly underrated web application hacking tools. The following steps will help you easily and systematically filter through thousands of lines of code in order to find sensitive information in page sources. Begin by opening your target page, and then open DevTools with F12 or `ctr-shift-I`. Adjust the DevTools window until you have enough space to work with. Select the Network tab and then refresh the page (`CTRL+r`).

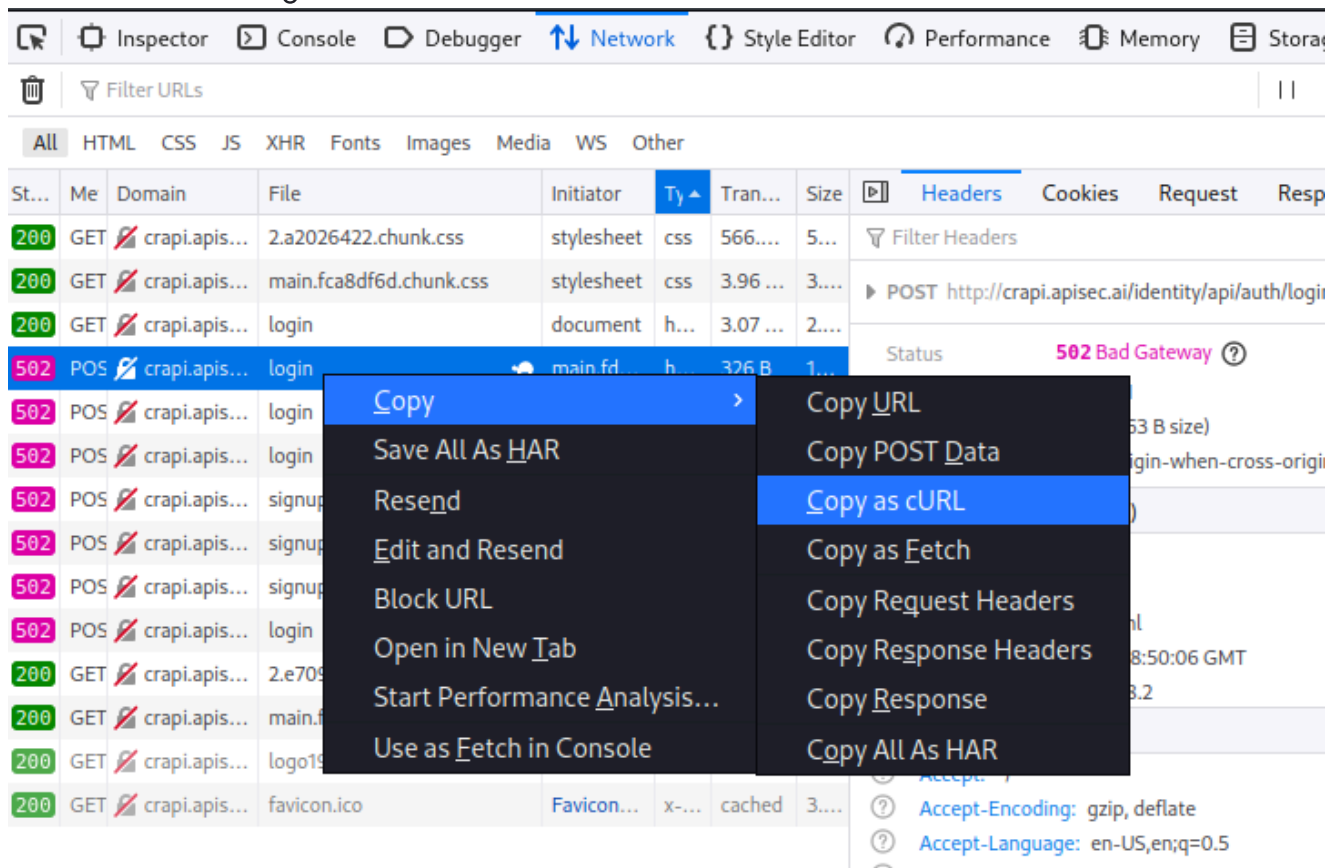


You can use the filter tool to search for any term you would like, such as "API", "v1", or "graphql". This is a quick way to find API endpoints in use. You can also leave the Devtools Network tab open while you perform actions on the web page. For example, let's check out what happens if we leave the DevTools open while we authenticate to crAPI. You should see a new request pop up. At this point, you can dive deeper into the request by right-clicking on one of the requests and selecting "Edit and Resend".

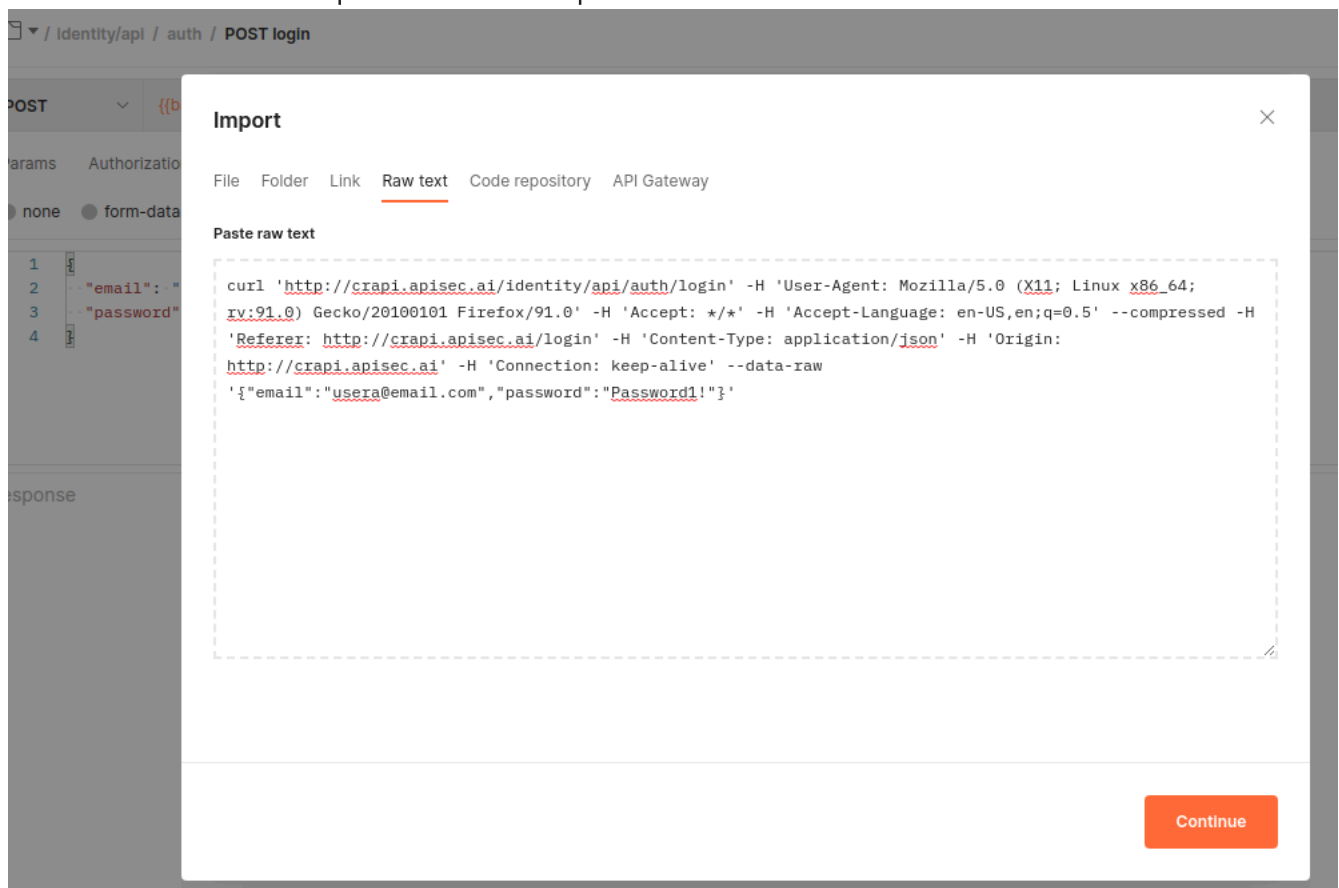




This will allow you to check out the request in the browser, edit the headers/request body, and send it back to the API provider. Although this is a great DevTools feature, you may want to move into a browser that was meant for interacting with APIs. You can use DevTools to migrate individual requests over to Postman using cURL.



Once you have copied the desired request, open Postman. Select Import and click on the "Raw text" tab. Paste in the cURL request and select import.



Once the request has been imported it will have all of the necessary headers and the request body necessary to make additional requests in Postman. This is a great way to quickly interact with an API and interact with a single API request. To automatically build out a more complete Postman Collection check out the next module which is on Reverse Engineering an API.

Reconnaissance is extremely important when testing APIs. Discovering API endpoints is a necessary first step when attacking APIs. Good recon also has the added benefit of potentially providing you with the keys to the castle in the form of API keys, passwords, tokens, and other useful information disclosures.

**Comments 0**

**🔒 Comments Locked**



# DISCORD

**Join the Discussion**

Looking for help, ask questions, chat with other API security pros? Join the discussion on Discord.

[Join Now](#)