PART II

BUILDING AN API TESTING LAB

4
YOUR API HACKING SYSTEM

This chapter will walk you through setting up your API hacking toolkit. We'll cover three especially useful tools for API hackers: Chrome DevTools, Burp Suite, and Postman.

In addition to exploring features included in the paid Burp Suite Pro version, I'll provide a list of tools that can compensate for the features missing from the free Burp Suite Community Edition, as well as several other tools useful for discovering and exploiting API vulnerabilities. At the end of this chapter, we'll walk through a lab in which you'll learn to use some of these tools to interact with our first APIs.

Kali Linux

Throughout this book, we'll run tools and labs using Kali, an open-source Debian-based distribution of Linux. Kali is built for penetration testing and comes with many useful tools already installed. You can download Kali at *https://www.kali.org/downloads*. Plenty of guides can walk you through setting up your hypervisor of choice and installing Kali onto it. I recommend using Null Byte's "How to Get Started with Kali Linux" or the tutorial at *https://www.kali.org/docs/installation*.

After your instance of Kali is set up, open a terminal and perform an update and upgrade:

```
$ sudo apt update
$ sudo apt full-upgrade -y
```

Next, install Git, Python 3, and Golang (Go), which you'll need to use some of the other tools in your hacking box:

```
$ sudo apt-get install git python3 golang
```

With these basics installed, you should be prepared to set up the remainder of the API hacking tools.

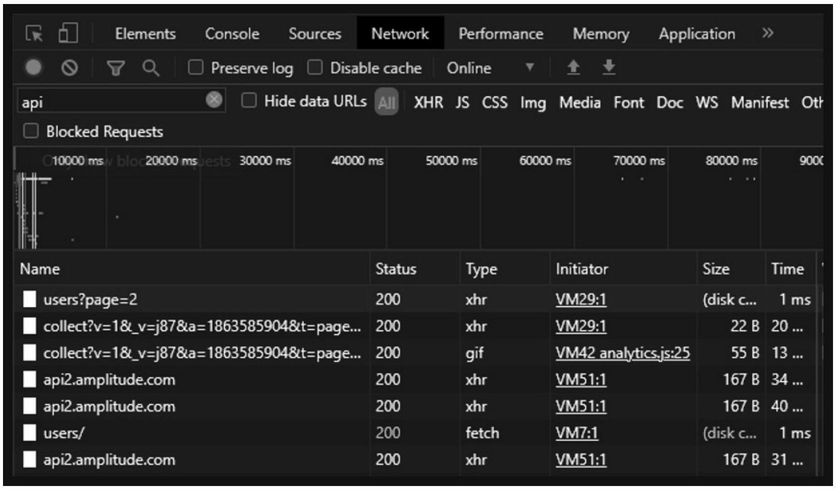
Analyzing Web Apps with DevTools

Chrome's DevTools is a suite of developer tools built into the Chrome browser that allows you to view what your web browser is running from a web developer's perspective. DevTools is an often-underrated resource, but it can be very useful for API hackers. We'll use it for our first interactions with target web applications to discover APIs; interact with web applications using the console; view headers, previews, and responses; and analyze web application source files.

To install Chrome, which includes DevTools, run the following commands:

```
$ sudo wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
$ sudo apt install ./google-chrome-stable_current_amd64.deb
```

You can launch Chrome through the command line with the `google -chrome` command. Once you have Chrome running, navigate to the URL you want to investigate and launch DevTools by using either CTRL-SHIFT-I or F12 or navigating to **Settings ▸ More Tools** and selecting the **Developer Tools** menu. Next, refresh your current page to update the information in the DevTools panels. You can do this by using the CTRL-R shortcut. In the Network panel, you should see the various resources requested from APIs (see Figure 4-1).



Name	Status	Type	Initiator	Size	Time
users?page=2	200	xhr	VM29:1	(disk c...	1 ms
collect?v=1&_v=j87&a=1863585904&t=page...	200	xhr	VM29:1	22 B	20 ...
collect?v=1&_v=j87&a=1863585904&t=page...	200	gif	VM42 analytics.js:25	55 B	13 ...
api2.amplitude.com	200	xhr	VM51:1	167 B	34 ...
api2.amplitude.com	200	xhr	VM51:1	167 B	40 ...
users/	200	fetch	VM7:1	(disk c...	1 ms
api2.amplitude.com	200	xhr	VM51:1	167 B	31 ...

Figure 4-1: The Chrome DevTools Network panel

Switch panels by selecting the desired tab at the top. The DevTools panel lists the functionality of the different table options. I've summarized these in Table 4-1.

Table 4-1: DevTools Panels

Panel	Function
Elements	Allows you to view the current page's CSS and Document Object Model (DOM), which enables you to inspect the HTML that constructs the web page.
Console	Provides you with alerts and lets you interact with the JavaScript debugger to alter the current web page.
Sources	Contains the directories that make up the web application and the content of the source files.
Network	Lists all the source file requests that make up the client's perspective of the web application.
Performance	Provides a way to record and analyze all the events that take place when loading a web page.
Memory	Lets you record and analyze how the browser is interacting with your system's memory.
Application	Provides you with the application manifest, storage items (like cookies and session information), cache, and background services.
Security	Provides insight regarding the transit encryption, source content origins, and certificate details.

When we first begin interacting with a web application, we'll usually start with the Network panel to get an overview of the resources that power the web application. In Figure 4-1, each of the items listed represents a request that was made for a specific resource. Using the Network panel, you can drill down into each request to see the request method that was used, the response status code, the headers, and the response body. To do this, click the name of the URL of interest once under the Name column. This will open up a panel on the right side of the DevTools. Now you can review the request that was made under the Headers tab and see how the server responded under the Response tab.

Diving deeper into the web application, you can use the Sources panel to inspect the source files being used in the app. In capture-the-flag (CTF) events (and occasionally in reality) you may find API keys or other hard-coded secrets here. The Sources panel comes equipped with strong search functionality that will help you easily discover the inner workings of the application.

The Console panel is useful for running and debugging the web page's JavaScript. You can use it to detect errors, view warnings, and execute commands. You will get an opportunity to use the Console panel in the lab in Chapter 6.

We will spend the majority of our time in the Console, Sources, and Network panels. However, the other panels can be useful as well. For example, the Performance panel is mainly used to improve a website's speed, but we could also use it to observe at what point a web application interacts with an API, as shown in Figure 4-2.

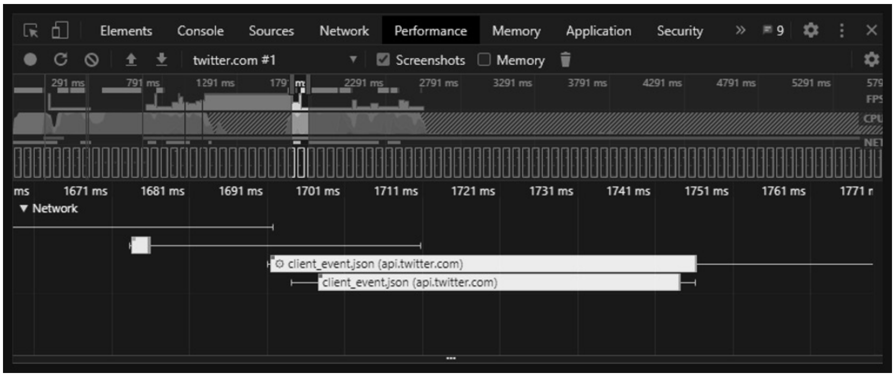


Figure 4-2: The DevTool's Performance tab showing the exact moment the Twitter application interacted with the Twitter API

In Figure 4-2 we see that, 1,700 milliseconds in, a client event triggered the Twitter application to interact with the API. As the client, we would then be able to correlate that event to an action we took on the page, such as authenticating to the web app, to know what the web application is using the API for. The more information we can gather before attacking an API, the better our odds will be at finding and exploiting vulnerabilities.

For more information about DevTools, check out the Google Developers documentation at *https://developers.google.com/web/tools/chrome-devtools.*

Capturing and Modifying Requests with Burp Suite

Burp Suite is a magnificent set of web application–testing tools developed and continuously improved on by PortSwigger. All web app cybersecurity professionals, bug bounty hunters, and API hackers should learn to use Burp, which allows you to capture API requests, spider web applications, fuzz APIs, and so much more.

Spidering, or *web crawling*, is a method that bots use to automatically detect the URL paths and resources of a host. Typically, spidering is done by scanning the HTML of web pages for hyperlinks. Spidering is a good way to get a basic idea of the contents of a web page, but it won't be able to find *hidden* paths, or the ones that do not have links found within web pages. To find hidden paths, we'll need to use a tool like Kiterunner that effectively performs directory brute-force attacks. In such an attack, an application will request various possible URL paths and validate whether they actually exist based on the host's responses.

As described by the OWASP community page on the topic, *fuzzing* is "the art of automatic bug finding." Using this attack technique, we'd send various types of input in HTTP requests, trying to find an input or payload that causes an application to respond in unexpected ways and reveal a vulnerability. For example, if you were attacking an API and discovered you could post data to the API provider, you could then attempt to send it various SQL commands. If the provider doesn't sanitize this input, there is a chance you could receive a response that indicates that a SQL database is in use.

Burp Suite Pro, the paid edition of Burp, provides all the features without restrictions, but if using the free Burp Suite Community Edition (CE) is your only option, you can make it work. However, once you've obtained a bug bounty reward or as soon as you can convince your employer, you should make the jump to Burp Suite Pro. This chapter includes a "Supplemental Tools" section that will help replace the functionality missing in Burp Suite CE.

Burp Suite CE is included standard with the latest version of Kali. If for whatever reason it is not installed, run the following:

```
$ sudo apt-get install burpsuite
```

NOTE *Burp Suite provides a full-featured 30-day trial version of Burp Suite Pro at* https://portswigger.net/requestfreetrial/pro. *For further instructions on using Burp Suite, visit* https://portswigger.net/burp/communitydownload.

In the following sections, we will prepare our API hacking rig to use Burp Suite, look at an overview of the various Burp modules, learn how to intercept HTTP requests, dive deeper into the Intruder module, and go over some of the sweet extensions you can use to enhance Burp Suite Pro.

Setting Up FoxyProxy

One of Burp Suite's key features is the ability to intercept HTTP requests. In other words, Burp Suite receives your requests before forwarding them to the server and then receives the server's responses before sending them to the browser, allowing you to view and interact with those requests and responses. For this feature to work, we'll need to regularly send requests from the browser to Burp Suite. This is done with the use of a web proxy. The proxy is a way for us to reroute web browser traffic to Burp before it is sent to the API provider. To simplify this process, we'll add a tool called FoxyProxy to our browsers to help us proxy traffic with a click of a button.

Web browsers have proxy functionality built in, but changing and updating these settings every time you want to use Burp would be a time-consuming pain. Instead, we'll use a browser add-on called FoxyProxy that lets you switch your proxy on and off with a simple click of a button. FoxyProxy is available for both Chrome and Firefox.

Follow these steps to install FoxyProxy:

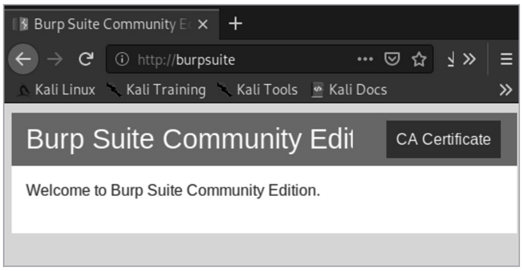
1. Navigate to your browser's add-on or plug-in store and search **FoxyProxy**.
2. Install FoxyProxy Standard and add it to your browser.
3. Click the fox icon at the top-right corner of your browser (next to your URL) and select **Options**.
4. Select **Proxies** ▶ **Add New Proxy** ▶ **Manual Proxy Configuration**.
5. Add **127.0.0.1** as the host IP address.
6. Update the port to **8080** (Burp Suite's default proxy settings).
7. Under the General tab, rename the proxy to **Hackz** (I will refer to this proxy setting throughout the labs).

Now you'll only need to click the browser add-on and select the proxy you want to use to send your traffic to Burp. When you've finished intercepting requests, you can turn the proxy off by selecting the Disable FoxyProxy option.

Adding the Burp Suite Certificate

HTTP Strict Transport Security (HSTS) is a common web application security policy that prevents Burp Suite from being able to intercept requests. Whether using Burp Suite CE or Burp Suite Pro, you will need to install Burp Suite's certificate authority (CA) certificate. To add this certificate, follow these steps:

1. Start Burp Suite.
2. Open your browser of choice.
3. Using FoxyProxy, select the Hackz proxy. Navigate to *http://burpsuite*, as seen in Figure 4-3, and click **CA Certificate**. This will initiate the download of the Burp Suite CA certificate.

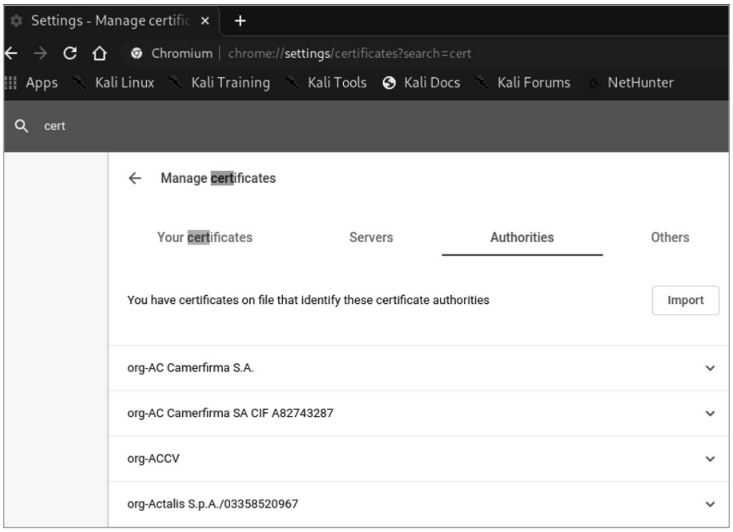


Burp Suite Community Edit

Welcome to Burp Suite Community Edition.

Figure 4-3: The landing page you should see when downloading Burp Suite's CA certificate

4. Save the certificate somewhere you can find it.
5. Open your browser and import the certificate. In Firefox, open **Preferences** and use the search bar to look up **certificates**. Import the certificate.
6. In Chrome, open **Settings**, use the search bar to look up **certificates**, select **More ▸ Manage Certificates ▸ Authorities**, and import the certificate (see Figure 4-4). If you do not see the certificate, you may need to expand the file type options to "DER" or "All files."



You have certificates on file that identify these certificate authorities

org-AC Camerfirma S.A.

org-AC Camerfirma SA CIF A82743287

org-ACCV

org-Actalis S.p.A./03358520967

Figure 4-4: The Chrome Certificate Manager with the Authorities tab selected

Now that you have the PortSwigger CA certificate added to your browser, you should be able to intercept traffic without experiencing issues.

Navigating Burp Suite

As you can see in Figure 4-5, at the top of Burp are 13 modules.

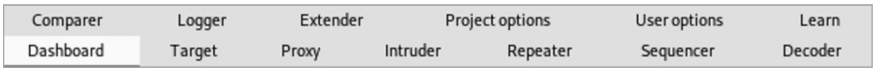


Figure 4-5: The Burp Suite modules

The *Dashboard* gives you an overview of the event log and scans you have run against your targets. The Dashboard is more useful in Burp Suite Pro than in CE because it will also display any issues detected during testing.

The *Proxy* tab is where we will begin capturing requests and responses from your web browser and Postman. The proxy we set up will send any web traffic destined for your browser here. We will typically choose to forward or drop captured traffic until we find the targeted site that we want to interact with. From Proxy we will forward the request or response to other modules for interaction and tampering.

In the *Target* tab, we can see a site's map and manage the targets we intend to attack. You can also use this tab to configure the scope of your testing by selecting the Scope tab and including or excluding URLs. Including URLs within scope will limit the URLs being attacked to only those you have authorization to attack.

While using the Target tab, you should be able to locate the *Site Map*, where you can see all the URLs Burp Suite has detected during your current Burp Suite session. As you perform scans, crawl, and proxy traffic, Burp Suite will start compiling a list of the target web applications and discovered directories. This is another place you can add or remove URLs from scope.

The *Intruder* tab is where we'll perform fuzzing and brute-force attacks against web applications. Once you've captured an HTTP request, you can forward it to Intruder, where you can select the exact parts of the request that you want to replace with the payload of your choice before sending it to the server.

The *Repeater* is a module that lets you make hands-on adjustments to HTTP requests, send them to the targeted web server, and analyze the content of the HTTP response.

The *Sequencer* tool will automatically send hundreds of requests and then perform an analysis of entropy to determine how random a given string is. We will primarily use this tool to analyze whether cookies, tokens, keys, and other parameters are actually random.

The *Decoder* is a quick way to encode and decode HTML, base64, ASCII hex, hexadecimal, octal, binary, and Gzip.

The *Comparer* can be used to compare different requests. Most often, you'll want to compare two similar requests and find the sections of the request that have been removed, added, and modified.

If Burp Suite is too bright for your hacker eyes, navigate to **User options ▸ Display** and change **Look and Feel** to **Darcula**. Within the User Options tab, you can also find additional connection configurations, TLS settings, and miscellaneous options to learn hotkey shortcuts or configure your own hotkeys. You can then save your preferred settings using Project Options, which allows you to save and load specific configurations you like to use per project.

Learn is an awesome set of resources to help you learn how to use Burp Suite. This tab contains video tutorials, the Burp Suite Support Center, a guided tour of Burp's features, and a link to the PortSwigger Web Security Academy. Definitely check these resources out if you are new to Burp!

Under the Dashboard you can find the Burp Suite Pro Scanner. *Scanner* is Burp Suite Pro's web application vulnerability scanner. It lets you automatically crawl web applications and scan for weaknesses.

The *Extender* is where we'll obtain and use Burp Suite extensions. Burp has an app store that allows you to find add-ons to simplify web app testing. Many extensions require Burp Suite Pro, but we will make the most of the free extensions to turn Burp into an API hacking powerhouse.

Intercepting Traffic

A Burp Suite session will usually begin with intercepting traffic. If you've set up FoxyProxy and the Burp Suite certificate correctly, the following process should work smoothly. You can use these instructions to intercept any HTTP traffic with Burp Suite:

1. Start Burp Suite and change the Intercept option to **Intercept is on** (see Figure 4-6).

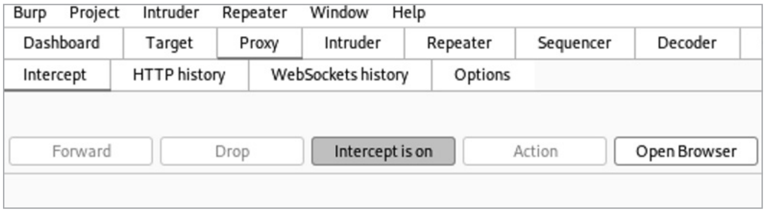


Figure 4-6: Intercept is on in Burp Suite.

2. In your browser, select the Hackz proxy using FoxyProxy and browse to your target, such as *https://twitter.com* (see Figure 4-7). This web page will not load in the browser because it was never sent to the server; instead, the request should be waiting for you in Burp Suite.

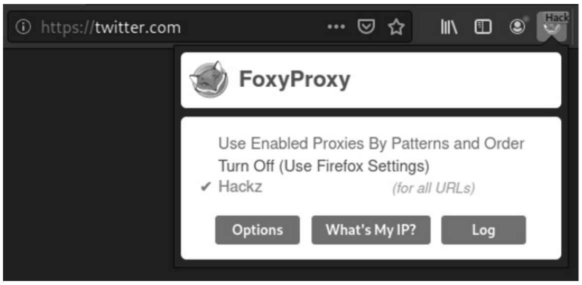
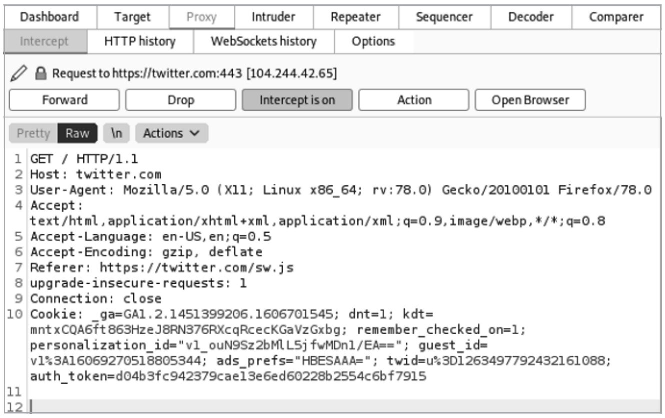

https://twitter.com
FoxyProxy
Use Enabled Proxies By Patterns and Order
Turn Off (Use Firefox Settings)
✔ Hackz
(for all URLs)
Options
What's My IP?
Log

Figure 4-7: The request to Twitter gets sent to Burp Suite via the Hackz proxy.

3. In Burp Suite, you should see something much like Figure 4-8. This should let you know that you've successfully intercepted an HTTP request.

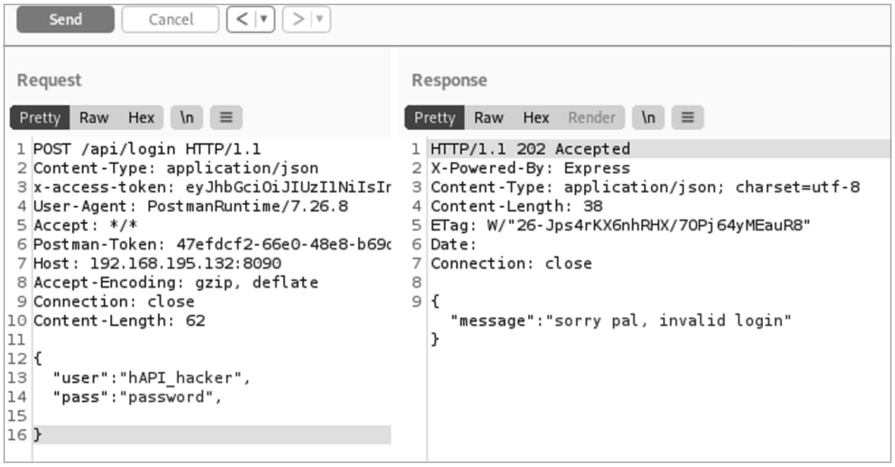


```
GET / HTTP/1.1
Host: twitter.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://twitter.com/sw.js
upgrade-insecure-requests: 1
Connection: close
Cookie: _ga=GA1.2.1451399206.1606701545; dnt=1; kdt=
mntxCQA6ft863HzeJ8RN376RXcqRcecKGaVzGxbg; remember_checked_on=1;
personalization_id="v1_ouN9Sz2bMlL5jfwMDn1/EA=="; guest_id=
v1%3A160692705188O5344; ads_prefs="HBESAAA="; twid=u%3D1263497792432161088;
auth_token=d04b3fc942379cae13e6ed60228b2554c6bf7915

```

Figure 4-8: An HTTP request to Twitter intercepted by Burp Suite

Once you've captured a request, you can select an action to perform with it, such as forwarding the intercepted request to the various Burp Suite modules. You perform actions by clicking the Action button above the request pane or by right-clicking the request window. You will then have the option to forward the request to one of the other modules, such as Repeater (see Figure 4-9).



```
POST /api/login HTTP/1.1
Content-Type: application/json
x-access-token: eyJhbGciOiJIUzI1NiIsIr
User-Agent: PostmanRuntime/7.26.8
Accept: */*
Postman-Token: 47efdcf2-66e0-48e8-b69c
Host: 192.168.195.132:8090
Accept-Encoding: gzip, deflate
Connection: close
Content-Length: 62

{
  "user":"hAPI_hacker",
  "pass":"password",

}
```

```
HTTP/1.1 202 Accepted
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 38
ETag: W/"26-Jps4rKX6nhRHX/7OPj64yMEauR8"
Date:
Connection: close

{
  "message":"sorry pal, invalid login"
}
```

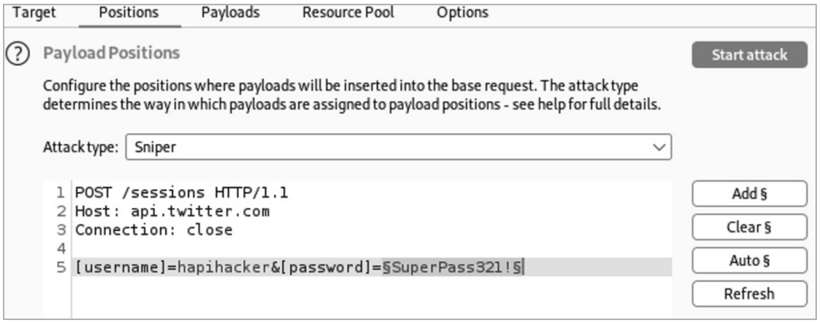
Figure 4-9: Burp Suite Repeater

The Repeater module is the best way to see how a web server responds to a single request. This is useful for seeing what sort of response you can expect to get from an API before initiating an attack. It's also helpful when you need to make minor changes to a request and want to see how the server responds.

Altering Requests with Intruder

We've already mentioned that Intruder is a web application fuzzing and scanning tool. It works by letting you create variables within an intercepted HTTP request, replace those variables with different sets of payloads, and send a series of requests to an API provider.

Any part of a captured HTTP request can be transformed into a variable, or *attack position*, by surrounding it with § symbols. Payloads can be anything from a wordlist to a set of numbers, symbols, and any other type of input that will help you test how an API provider will respond. For example, in Figure 4-10, we've selected the password as the attack position, as indicated by the § symbols.



Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

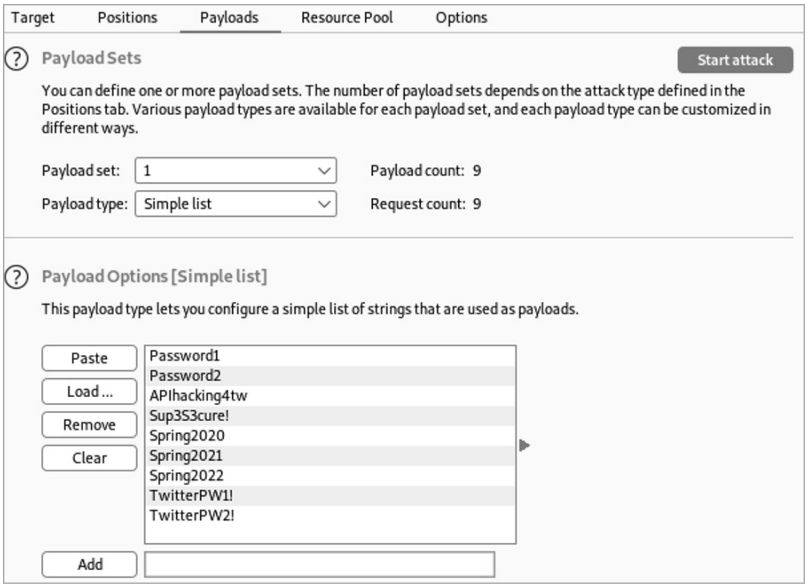
Attack type: Sniper

```
POST /sessions HTTP/1.1
Host: api.twitter.com
Connection: close

[username]=hapihacker&[password]=§SuperPass321!§
```

Figure 4-10: An Intruder attack against api.twitter.com

This means that `SuperPass321!` will be replaced with values from the list of strings found in Payloads. Navigate to the Payloads tab to see these strings, shown in Figure 4-11.



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1   Payload count: 9

Payload type: Simple list   Request count: 9

Payload Options [Simple list]

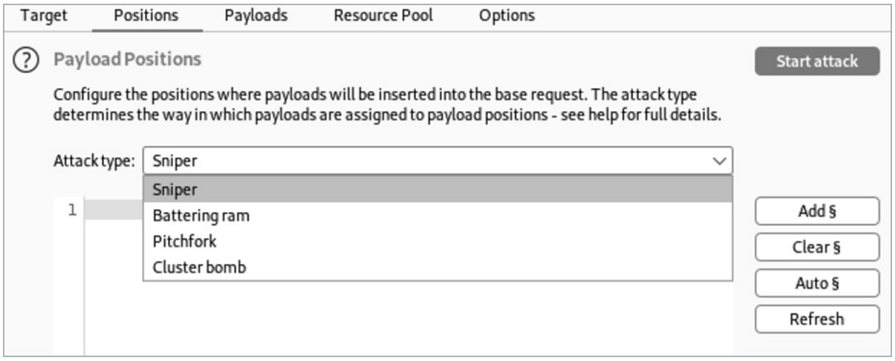
This payload type lets you configure a simple list of strings that are used as payloads.

```
Password1
Password2
APIhacking4tw
Sup3S3cure!
Spring2020
Spring2021
Spring2022
TwitterPW1!
TwitterPW2!
```

Figure 4-11: The Intruder Payloads with a list of passwords

Based on the payload list shown here, Intruder will perform one request per payload listed for a total of nine requests. When an attack is started, each of the strings under Payload Options will replace `SuperPass123!` in turn and generate a request to the API provider.

The Intruder attack types determine how the payloads are processed. As you can see in Figure 4-12, there are four different attack types: sniper, battering ram, pitchfork, and cluster bomb.


Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

Figure 4-12: The Intruder attack types

Sniper is the simplest attack type; it replaces the added attack position with a string provided from a single set of payloads. A sniper attack is limited to using a single payload, but it can have several attack positions. A sniper attack will replace one attack position per request, cycling through the different attack positions in each request. If you were attacking three different variables with a single payload, it would look something like this:

```
§Variable1§, §variable2§, §variable3§
Request 1:   Payload1, variable2, variable3
Request 2:   Variable1, payload1, variable3
Request 3:   Variable1, variable2, payload1
```

Battering ram is like the sniper attack in that it also uses one payload, but it will use that payload across all attack positions in a request. If you were testing for SQL injection across several input positions within a request, you could fuzz them all simultaneously with battering ram.

Pitchfork is used for testing multiple payload combinations at the same time. For example, if you have a list of leaked usernames and password combinations, you could use two payloads together to test whether any of the credentials were used with the application being tested. However, this attack doesn't try out different combinations of payloads; it will only cycle through the payload sets like this: *user1:pass1*, *user2:pass2*, *user3:pass3*.

Cluster bomb will cycle through all possible combinations of the payloads provided. If you provide two usernames and three passwords, the payloads would be used in the following pairs: *user1:pass1*, *user1:pass2*, *user1:pass3*, *user2:pass1*, *user2:pass2*, *user2:pass3*.

The attack type to use depends on your situation. If you're fuzzing a single attack position, use sniper. If you're fuzzing several attack positions at once, use battering ram. When you need to test set combinations of payloads, use pitchfork. For password-spraying efforts, use cluster bomb.

Intruder should help you find API vulnerabilities such as broken object level authorization, excessive data exposure, broken authentication, broken function level authorization, mass assignment, injection, and improper assets management. Intruder is essentially a smart fuzzing tool that provides a list of results containing the individual requests and responses. You can interact with the request you'd like to fuzz and replace the attack position with the input of your choice. These API vulnerabilities are typically discovered by sending the right payload to the right location.

For example, if an API were vulnerable to authorization attacks like BOLA, we would be able to replace requested resource IDs with a payload containing a list of possible resource IDs. We could then start the attack with Intruder, which would make all the requests and provide us with a list of results to review. I will cover API fuzzing in Chapter 9 and API authorization attacks in Chapter 10.

EXTENDING THE POWER OF BURP SUITE

One of the major benefits of Burp Suite is that you can install custom extensions. These extensions can help you shape Burp Suite into the ultimate API hacking tool. To install extensions, use the search bar to find the one you're looking for and then click the **Install** button. Some extensions require additional resources and have more complex installation requirements. Make sure you follow the install instructions for each extension. I recommend adding the following ones.

AUTORIZE

Autorize is an extension that helps automate authorization testing, particularly for BOLA vulnerabilities. You can add the tokens of UserA and UserB accounts and then perform a bunch of actions to create and interact with resources as UserA. Also, Autorize can automatically attempt to interact with UserA's resources with the UserB account. Autorize will highlight any interesting requests that may be vulnerable to BOLA.

JSON WEB TOKENS

The JSON Web Tokens extension helps you dissect and attack JSON Web Tokens. We will use this extension to perform authorization attacks later in Chapter 8.

InQL SCANNER

InQL is an extension that will aid us in our attacks against GraphQL APIs. We will make the most out of this extension in Chapter 14.

(continued)

IP ROTATE

IP Rotate allows you to alter the IP address you are attacking from to indicate different cloud hosts in different regions. This is extremely useful against API providers that simply block attacks based on IP address.

BYPASS WAF

The WAF Bypass extension adds some basic headers to your requests in order to bypass some web application firewalls (WAFs). Some WAFs can be tricked by the inclusion of certain IP headers in the request. WAF Bypass saves you from manually adding headers such as `X-Originating-IP`, `X-Forwarded-For`, `X-Remote-IP`, and `X-Remote-Addr`. These headers normally include an IP address, and you can specify an address that you believe to be permitted, such as the target's external IP address (127.0.0.1) or an address you suspect to be trusted.

In the lab at the end of this chapter, I will walk you through interacting with an API, capturing the traffic with Burp Suite, and using Intruder to discover a list of existing user accounts. To learn more about Burp Suite, visit the PortSwigger WebSecurity Academy at *https://portswigger.net/web-security* or consult the Burp Suite documentation at *https://portswigger.net/burp/documentation*.

Crafting API Requests in Postman, an API Browser

We'll use Postman to help us craft API requests and visualize responses. You can think of Postman as a web browser built for interacting with APIs. Originally designed as a REST API client, it now has all sorts of capabilities for interacting with REST, SOAP, and GraphQL. The application is packed with features for creating HTTP requests, receiving responses, scripting, chaining requests together, creating automated testing, and managing API documentation.

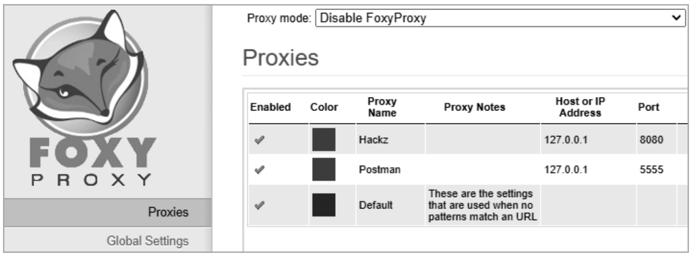
We'll be using Postman as our browser of choice for sending API requests to a server, rather than defaulting to Firefox or Chrome. This section covers the Postman features that matter the most and includes instructions for using the Postman request builder, an overview of working with collections, and some basics around building request tests. Later in this chapter, we will configure Postman to work seamlessly with Burp Suite.

To set up Postman on Kali, open your terminal and enter the following commands:

```
$ sudo wget https://dl.pstmn.io/download/latest/linux64 -O postman-linux-x64.tar.gz
$ sudo tar -xvzf postman-linux-x64.tar.gz -C /opt
$ sudo ln -s /opt/Postman/Postman /usr/bin/postman
```

If everything has gone as planned, you should be able to launch Postman by entering `postman` in your terminal. Sign up for a free account using an email address, username, and password. Postman uses accounts for collaboration and to synchronize information across devices. Alternatively, you can skip the login screen by clicking the **Skip signing in and take me straight to the app** button.

Next, you'll need to go through the FoxyProxy setup process a second time (refer to the "Setting Up FoxyProxy" section earlier in this chapter) so that Postman can intercept requests. Return to step 4 and add a new proxy. Add the same host IP address, **127.0.0.1**, and set the port to **5555**, the default port for Postman's proxy. Update the name of the proxy under the General tab to **Postman** and save. Your FoxyProxy tab should now resemble Figure 4-13.



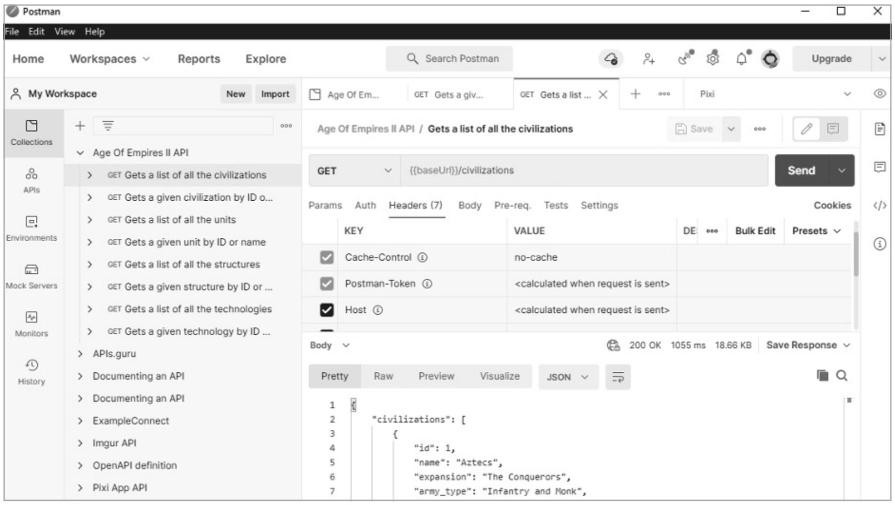
Proxy mode: Disable FoxyProxy

Proxies

Enabled	Color	Proxy Name	Proxy Notes	Host or IP Address	Port
✓	■	Hackz		127.0.0.1	8080
✓	■	Postman		127.0.0.1	5555
✓	■	Default	These are the settings that are used when no patterns match an URL		

Figure 4-13: FoxyProxy with the Hackz and Postman proxies set up

From the launchpad, open a new tab just like you would in any other browser by clicking the new tab button (+) or using the CTRL-T shortcut. As you can see in Figure 4-14, Postman's interface can be a little overwhelming if you aren't familiar with it.



Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Upgrade
My Workspace New Import
Collections APIs Environments Mock Servers Monitors History
Age Of Empires II API
GET Gets a list of all the civilizations
GET Gets a given civilization by ID o...
GET Gets a list of all the units
GET Gets a given unit by ID or name
GET Gets a list of all the structures
GET Gets a given structure by ID or ...
GET Gets a list of all the technologies
GET Gets a given technology by ID ...
APIs.guru
Documenting an API
Documenting an API
ExampleConnect
Imgur API
OpenAPI definition
Pixi App API
Age Of Em... GET Gets a giv... GET Gets a list ...
Pixi
Age Of Empires II API / Gets a list of all the civilizations
Save
GET {{baseUrl}}/civilizations Send
Params Auth Headers (7) Body Pre-req. Tests Settings Cookies
Bulk Edit Presets
Body 200 OK 1055 ms 18.66 KB Save Response
Pretty Raw Preview Visualize JSON

KEY	VALUE
✓ Cache-Control	no-cache
✓ Postman-Token	<calculated when request is sent>
✓ Host	<calculated when request is sent>

```
{
    "civilizations": [
        {
            "id": 1,
            "name": "Aztecs",
            "expansion": "The Conquerors",
            "army_type": "Infantry and Monk",
```

Figure 4-14: The main landing page of Postman with a response from an API collection

Let's start by discussing the request builder, which you'll see when you open a new tab.

The Request Builder

The request builder, shown in Figure 4-15, is where you can craft each request by adding parameters, authorization headers, and so on.

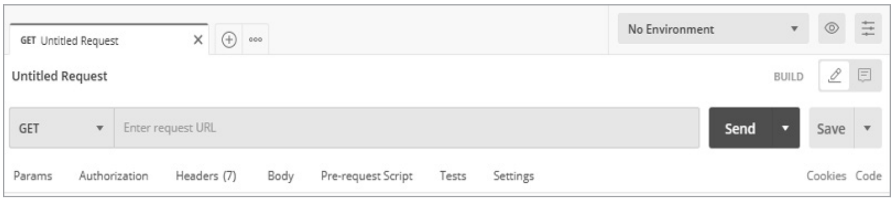


Figure 4-15: The Postman request builder

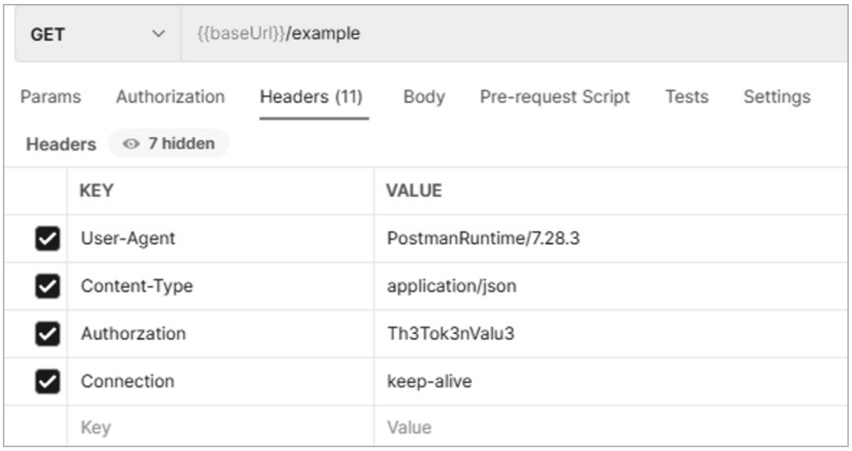
The request builder contains several tabs useful for precisely constructing the parameters, headers, and body of a request. The *Params* tab is where you can add query and path parameters to a request. Essentially, this allows you to enter in various key/value pairs along with a description of those parameters. A great feature of Postman is that you can leverage the power of variables when creating your requests. If you import an API and it contains a variable like *:company* in *http://example.com/**:company**/profile*, Postman will automatically detect this and allow you to update the variable to a different value, such as the actual company name. We'll discuss collections and environments later in this section.

The *Authorization* tab includes many standard forms of authorization headers for you to include in your request. If you've saved a token in an environment, you can select the type of token and use the variable's name to include it. By hovering your mouse over a variable name, you can see the associated credentials. Several authorization options are available under the Type field that will help you automatically format the authorization header. Authorization types include several expected options such as no auth, API key, Bearer Token, and Basic Auth. In addition, you could use the authorization that is set for the entire collection by selecting **inherit auth from parent**.

The *Headers* tab includes the key and value pairs required for certain HTTP requests. Postman has some built-in functionality to automatically create necessary headers and to suggest common headers with preset options.

In Postman, values for parameters, headers, and parts of body work can be added by entering information within the Key column and the corresponding Value column (see Figure 4-16). Several headers will automatically be created, but you can add your own headers when necessary.

Within the keys and values, you also have the ability to use collection variables and environmental variables. (We'll cover collections later in this section.) For example, we've represented the value for the password key using the variable name `{admin_creds}`.



	KEY	VALUE
✓	User-Agent	PostmanRuntime/7.28.3
✓	Content-Type	application/json
✓	Authorzation	Th3Tok3nValu3
✓	Connection	keep-alive
	Key	Value

Figure 4-16: Postman key and value headers

The request builder can also run pre-request scripts, which can chain together different requests that depend on each other. For example, if request 1 issues a resource value that is needed for the following request, you can script that resource value to automatically be added to request 2.

Within Postman's request builder, you can use several panels to craft proper API requests and review responses. Once you've sent a request, the response will show up in the response panel (see Figure 4-17).

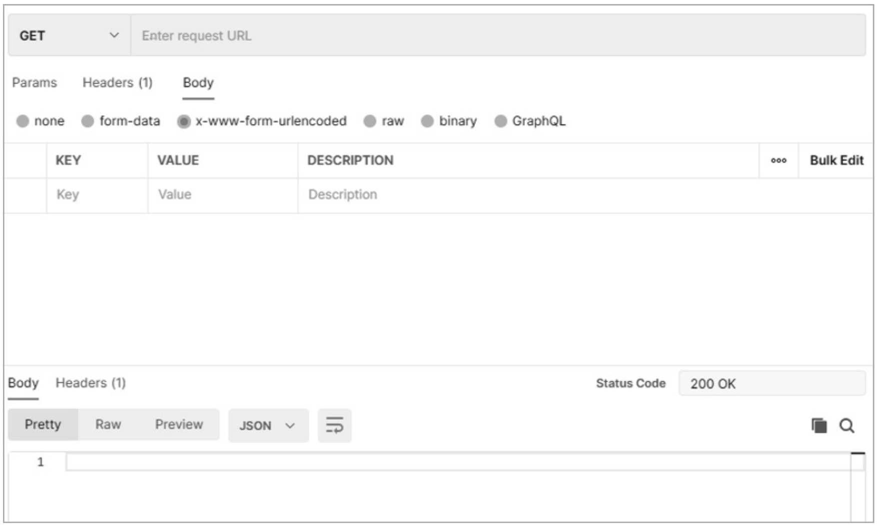


Figure 4-17: The Postman request and response panels

You can set the response panel either to the right or below the request panel. By pressing CTRL-ALT-V, you can switch the request and response panels between single-pane and split-pane views.

In Table 4-2, I have separated the items into the request panels and the response panels.

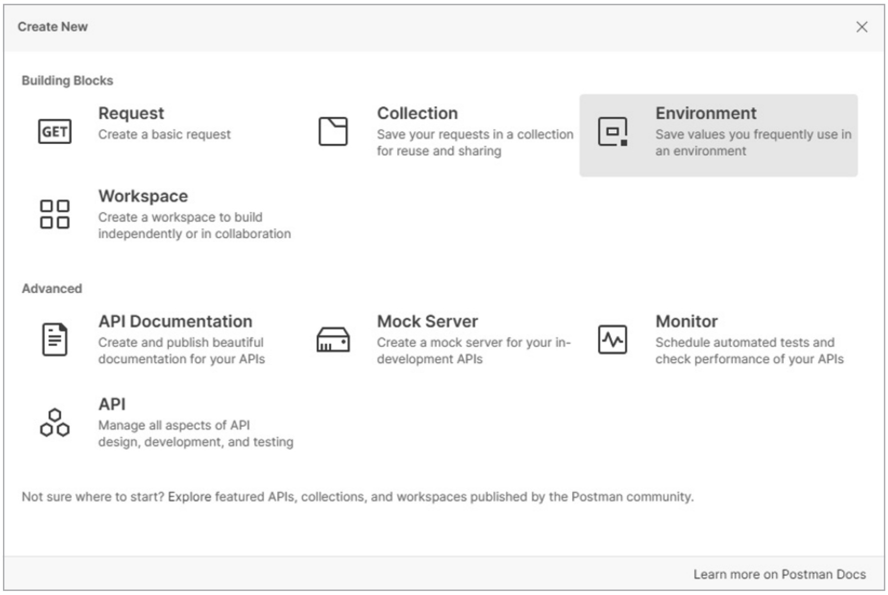
Table 4-2: Request Builder Panels

Panel	Purpose
Request	
HTTP request method	The request method is found to the left of the request URL bar (at the top left of Figure 4-17 where there is a drop-down menu for GET). The options include all the standard requests: GET, POST, PUT, PATCH, DELETE, HEAD, and OPTIONS. It also includes several other request methods such as COPY, LINK, UNLINK, PURGE, LOCK, UNLOCK, PROPFIND, and VIEW.
Body	In Figure 4-17, this is the third tab in the request pane. This allows for adding body data to the request, which is primarily used for adding or updating data when using PUT, POST, or PATCH.
Body options	Body options are the format of the response. These are found below the Body tab when it is selected. The options currently include none, form-data, x-www-formurlencoded, raw, binary, and GraphQL. These options let you view response data in various forms.
Pre-request script	JavaScript-based scripts that can be added and executed before a request is sent. This can be used to create variables, help troubleshoot errors, and change request parameters.
Test	This space allows for writing JavaScript-based tests used to analyze and test the API response. This is used to make sure the API responses are functioning as anticipated.
Settings	Various settings for how Postman will handle requests.
Response	
Response body	The body of the HTTP response. If Postman were a typical web browser, this would be the main window to view the requested information.
Cookies	This shows all the cookies, if any, included with the HTTP response. This tab will include information about the cookie type, cookie value, path, expiration, and cookie security flags.
Headers	This is where all the HTTP response headers are located.
Test results	If you created any tests for your request, this is where you can view the results of those tests.

Environments

An *environment* provides a way to store and use the same variables across APIs. An *environmental variable* is a value that will replace a variable across an environment. For example, say you're attacking a production API but discover a *test* version of the production API as well; you'll likely want to use an environment to share values between your requests to the two APIs. After all, there is a chance the production and test APIs share values such as API tokens, URL paths, and resource IDs.

To create environmental variables, find **Environment** at the top right of the request builder (the drop-down menu that says "No Environment" by default) and then press CTRL-N to bring up the **Create New** panel and select **Environment**, as shown in Figure 4-18.


Create New

Building Blocks

Request
Create a basic request

Collection
Save your requests in a collection for reuse and sharing

Environment
Save values you frequently use in an environment

Workspace
Create a workspace to build independently or in collaboration

Advanced

API Documentation
Create and publish beautiful documentation for your APIs

Mock Server
Create a mock server for your in-development APIs

Monitor
Schedule automated tests and check performance of your APIs

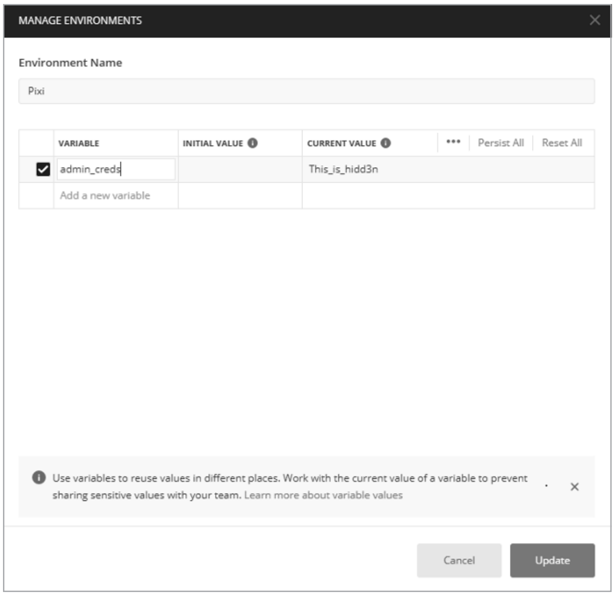
API
Manage all aspects of API design, development, and testing

Not sure where to start? Explore featured APIs, collections, and workspaces published by the Postman community.

Learn more on Postman Docs

Figure 4-18: The Create New panel in Postman

You can give an environment variable both an initial value and a current value (see Figure 4-19). An *initial value* will be shared if you share your Postman environment with a team, whereas a current value is not shared and is only stored locally. For example, if you have a private key, you can store the private key as the current value. Then you will be able to use the variable in places where you would have to paste the private key.


MANAGE ENVIRONMENTS

Environment Name

Pixi

	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	admin_creds		This_is_hidd3n
	Add a new variable		

Use variables to reuse values in different places. Work with the current value of a variable to prevent sharing sensitive values with your team. Learn more about variable values

Figure 4-19: The Manage Environments window in Postman showing the variable `admin_creds` *with a current value of* `This_is_hidd3n`

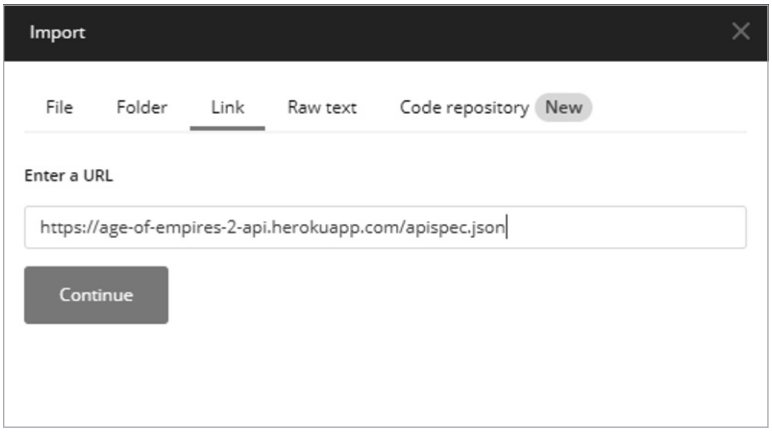
Collections

Collections are groups of API requests that can be imported into Postman. If an API provider offers a collection, you won't have to physically type in every single request. Instead, you can just import its collection. The best way to understand this functionality is to download a public API collection to your Postman from *https://www.postman.com/explore/collections*. For examples throughout this section, I will be referencing the Age of Empires II collection.

The Import button lets you import collections, environments, and API specifications. Currently, Postman supports OpenAPI 3.0, RAML 0.8, RAML 1.0, GraphQL, cURL, WADL, Swagger 1.2, Swagger 2.0, Runscope, and DHC. You can make your testing quite a bit easier if you can import your target API specification. Doing this will save you the time of having to craft all the API requests by hand.

Collections, environments, and specifications can all be imported as a file, folder, link, or raw test or through linking your GitHub account. For example, you can import the API for the classic PC game *Age of Empires II* from *https://age-of-empires-2-api.herokuapp.com/apispec.json* as follows:

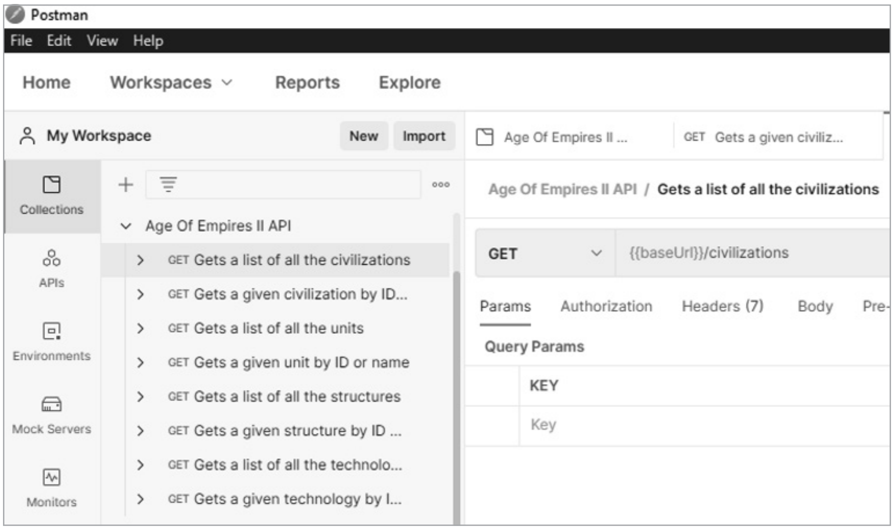
1. Click the **Import** button found at the top left of Postman.
2. Select the **Link** tab (see Figure 4-20).
3. Paste the URL to the API specification and click **Continue**.
4. On the Confirm Your Import screen, click **Import**.



https://age-of-empires-2-api.herokuapp.com/apispec.json

Figure 4-20: Importing an API specification in Postman using the Link tab in the Import panel

Once this is complete, you should have the Age of Empires II collection saved in Postman. Now test it out. Select one of the requests in the collection shown in Figure 4-21 and click **Send**.



Age Of Empires II API
- GET Gets a list of all the civilizations
- GET Gets a given civilization by ID...
- GET Gets a list of all the units
- GET Gets a given unit by ID or name
- GET Gets a list of all the structures
- GET Gets a given structure by ID ...
- GET Gets a list of all the technolo...
- GET Gets a given technology by I...

Age Of Empires II API / Gets a list of all the civilizations

GET {{baseUrl}}/civilizations

Figure 4-21: The Collections sidebar with the imported Age of Empires II API GET requests

For the request to work, you might have to first check the collection's variables to make sure they're set to the correct values. To see a collection's variables, you will need to navigate to the Edit Collection window by selecting **Edit** within the **View More Actions** button (represented by three circles, as shown in Figure 4-22).

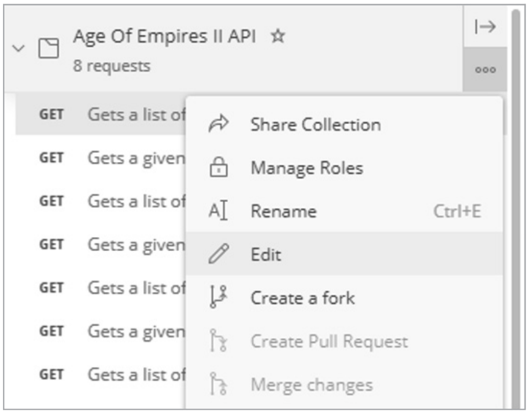
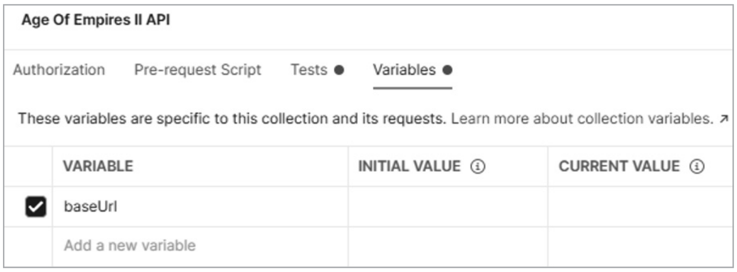


Figure 4-22: Editing a collection within Postman

Once you're in the Edit Collection window, select **Variables**, as shown in Figure 4-23.

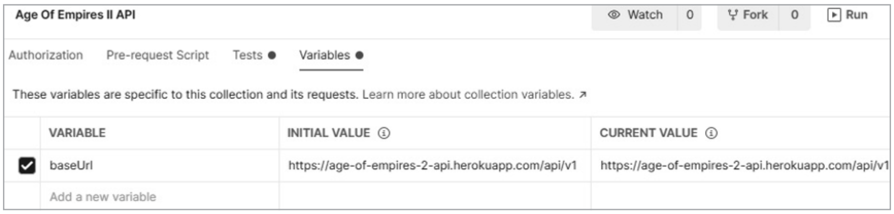


These variables are specific to this collection and its requests. Learn more about collection variables. ↗

	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	baseUrl		
	Add a new variable		

Figure 4-23: The Age of Empires II API collection variables

For example, the Age of Empires II API collection uses the variable `{{baseUrl}}`. The problem with the current `{{baseUrl}}` is that there are no values. We need to update this variable to the full URL of the public API, *https://age-of-empires-2-api.herokuapp.com/api/v1*. Add the full URL and click **Save** to update your changes (see Figure 4-24).

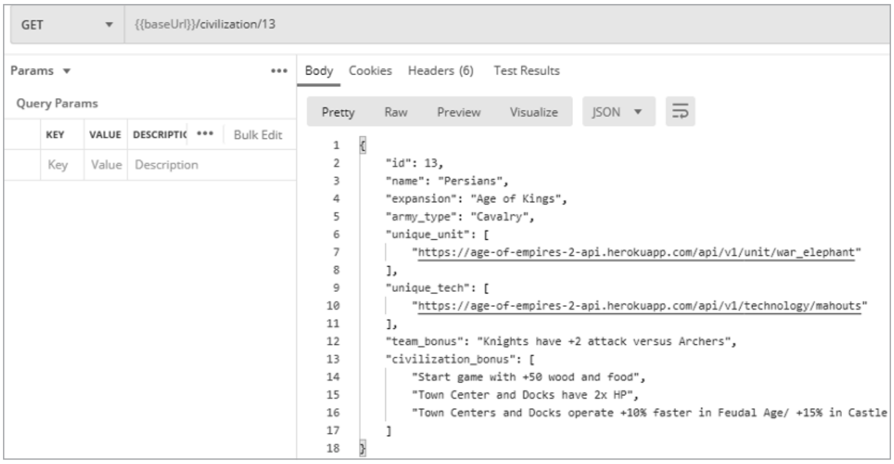


These variables are specific to this collection and its requests. Learn more about collection variables. ↗

	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	baseUrl	https://age-of-empires-2-api.herokuapp.com/api/v1	https://age-of-empires-2-api.herokuapp.com/api/v1
	Add a new variable		

Figure 4-24: The updated `baseURL` variable

Now that the variable is updated, you can choose one of the requests and click **Send**. If you are successful, you should receive a response similar to that shown in Figure 4-25.



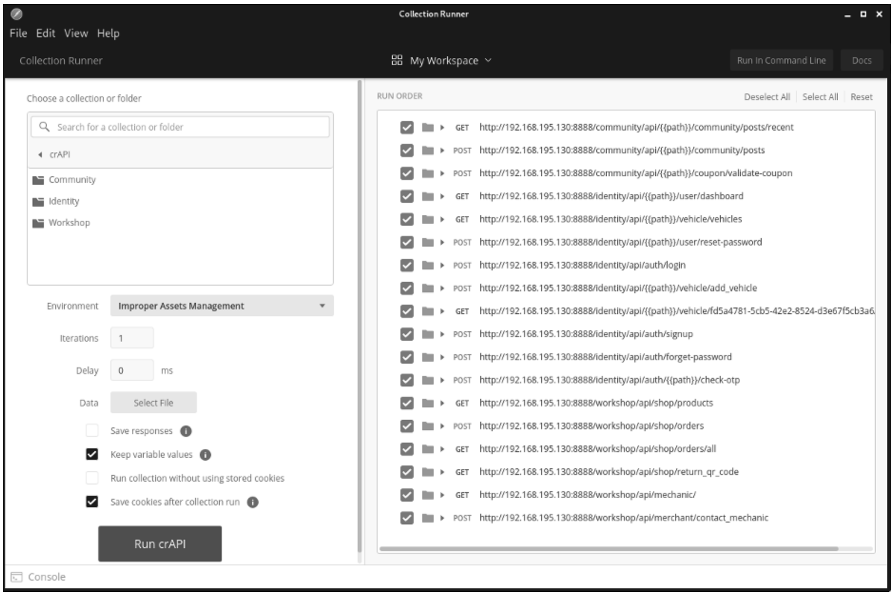
```
{
    "id": 13,
    "name": "Persians",
    "expansion": "Age of Kings",
    "army_type": "Cavalry",
    "unique_unit": [
        "https://age-of-empires-2-api.herokuapp.com/api/v1/unit/war_elephant"
    ],
    "unique_tech": [
        "https://age-of-empires-2-api.herokuapp.com/api/v1/technology/mahouts"
    ],
    "team_bonus": "Knights have +2 attack versus Archers",
    "civilization_bonus": [
        "Start game with +50 wood and food",
        "Town Center and Docks have 2x HP",
        "Town Centers and Docks operate +10% faster in Feudal Age/ +15% in Castle
    ]
}
```

Figure 4-25: Successfully using the Age of Empires II API collection in Postman

Whenever you import a collection and run into errors, you can use this process to troubleshoot the collection's variables. Also be sure to check that you haven't omitted any authorization requirements.

The Collection Runner

The Collection Runner allows you to run all the saved requests in a collection (see Figure 4-26). You can select the collection you want to run, the environment you want to pair it with, how many times you want to run the collection, and a delay in case there are rate-limiting requirements.



Environment: Improper Assets Management
Iterations: 1
Delay: 0 ms
Data
☐ Save responses
☑ Keep variable values
☐ Run collection without using stored cookies
☑ Save cookies after collection run

```
GET  http://192.168.195.130:8888/community/api/{{path}}/community/posts/recent
POST http://192.168.195.130:8888/community/api/{{path}}/community/posts
POST http://192.168.195.130:8888/community/api/{{path}}/coupon/validate-coupon
GET  http://192.168.195.130:8888/identity/api/{{path}}/user/dashboard
GET  http://192.168.195.130:8888/identity/api/{{path}}/vehicle/vehicles
POST http://192.168.195.130:8888/identity/api/{{path}}/user/reset-password
POST http://192.168.195.130:8888/identity/api/auth/login
POST http://192.168.195.130:8888/identity/api/{{path}}/vehicle/add_vehicle
GET  http://192.168.195.130:8888/identity/api/{{path}}/vehicle/fd5a4781-5cb5-42e2-8524-d3e67f5cb3a6
POST http://192.168.195.130:8888/identity/api/auth/signup
POST http://192.168.195.130:8888/identity/api/auth/forget-password
POST http://192.168.195.130:8888/identity/api/auth/{{path}}/check-otp
GET  http://192.168.195.130:8888/workshop/api/shop/products
POST http://192.168.195.130:8888/workshop/api/shop/orders
GET  http://192.168.195.130:8888/workshop/api/shop/orders/all
GET  http://192.168.195.130:8888/workshop/api/shop/return_qr_code
GET  http://192.168.195.130:8888/workshop/api/mechanic/
POST http://192.168.195.130:8888/workshop/api/merchant/contact_mechanic
```

Figure 4-26: The Postman Collection Runner

The requests can also be put into a specific order. Once the Collection Runner has run, you can review the Run Summary to see how each request was handled. For instance, if I open the Collection Runner, select Twitter API v2, and run the Collection Runner, I can see an overview of all API requests in that collection.

Code Snippets

In addition to the panels, you should also be aware of the code snippets feature. At the top-right of the request pane, you'll see a Code button. This button can be used to translate the built request into many different formats, including cURL, Go, HTTP, JavaScript, NodeJS, PHP, and Python. This is a helpful feature when we craft a request with Postman and then need to pivot to another tool. You can craft a complicated API request in Postman, generate a cURL request, and then use that with other command line tools.

The Tests Panel

The Tests panel allows you to create scripts that will be run against responses to your requests. If you are not a programmer, you will appreciate that Postman has made prebuilt code snippets available on the right side of the Tests panel. You can easily build a test by finding a prebuilt code snippet, clicking it, and adjusting the test to fit your testing needs. I suggest checking out the following snippets:

- `Status code: Code is 200`
- `Response time is less than 200ms`
- `Response body: contains string`

These JavaScript code snippets are fairly straightforward. For instance, the test for `Status code: Code is 200` is as follows:

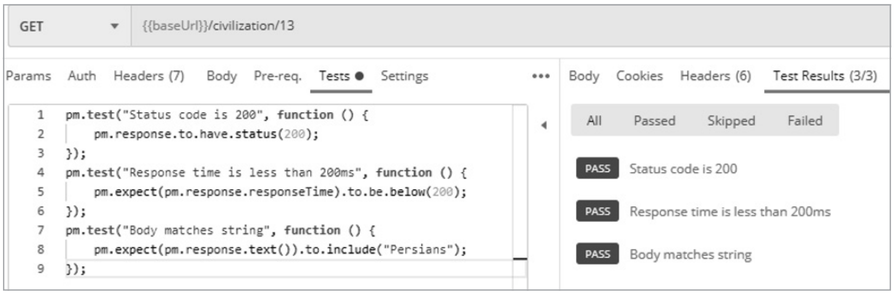
```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

You can see that the name of the test that will be displayed in the test results is "Status code is 200." The function is checking to make sure the Postman response has the status 200. We can easily adjust JavaScript to check for any status code by simply updating the `(200)` to our desired status code and changing the test name to fit. For example, if we wanted to check for the status code 400, we could change the code as follows:

```
pm.test("Status code is 400", function () {
    pm.response.to.have.status(400);
});
```

It's as simple as that! You really don't have to be a programmer to understand these JavaScript code snippets.

Figure 4-27 shows a series of tests included with the API request to the AOE2 public API. The tests include a check for a 200 status code, less than 200 ms latency, and "Persians" within the response string.



```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});
pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include("Persians");
});
```

```
PASS  Status code is 200
PASS  Response time is less than 200ms
PASS  Body matches string
```

Figure 4-27: AOE2 public API tests

After your tests are configured, you can check the Test Results tab of a response to see if the tests succeeded or failed. A good practice with creating tests is to make sure the tests fail. Tests are only effective if they pass and fail when they are supposed to. Therefore, send a request that would create conditions you would expect to pass or fail the test to ensure it is functioning properly. For more information about creating test scripts, check out the Postman documentation (*https://learning.postman.com/docs/writing-scripts/test-scripts*).

You now have many other options to explore in Postman. Like Burp Suite, Postman has a Learning Center (*https://learning.postman.com*) for online resources for those who want to develop a deeper understanding of the software. Alternatively, if you would like to review the Postman documentation, you can find it at *https://learning.postman.com/docs/getting-started/introduction*.

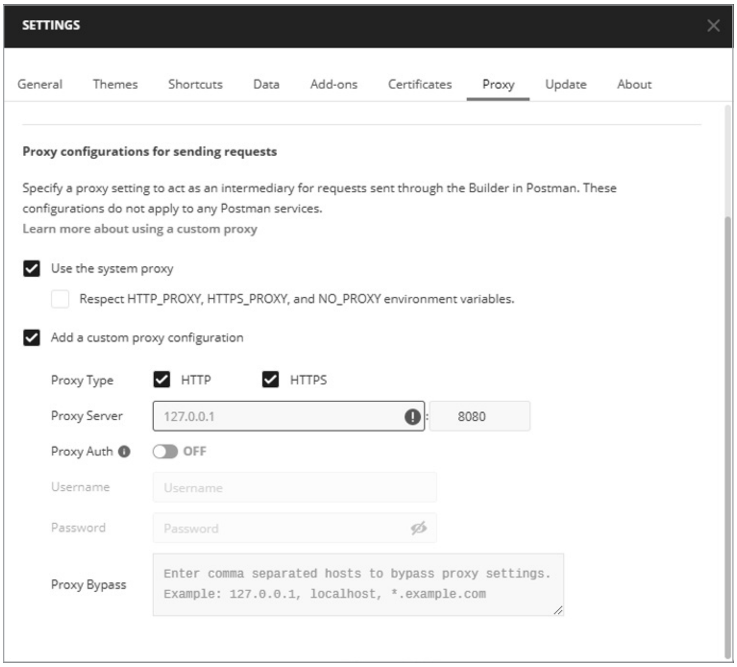
Configuring Postman to Work with Burp Suite

Postman is useful for interacting with APIs, and Burp Suite is a powerhouse for web application testing. If you combine these applications, you can configure and test an API in Postman and then proxy the traffic over to Burp Suite to brute-force directories, tamper with parameters, and fuzz all the things.

As when you set up FoxyProxy, you'll need to configure the Postman proxy to send traffic over to Burp Suite using the following steps (see Figure 4-28):

1. Open Postman settings by pressing CTRL-, (comma) or navigating to **File ▸ Settings**.
2. Click the **Proxy** tab.
3. Click the checkbox for adding a custom proxy configuration.
4. Make sure to set the proxy server to **127.0.0.1**.
5. Set the proxy server port to **8080**.
6. Select the **General** tab and turn SSL certificate verification **Off**.

7. In Burp Suite, select the **Proxy** tab.
8. Click the button to turn Intercept **On**.



Proxy configurations for sending requests

Specify a proxy setting to act as an intermediary for requests sent through the Builder in Postman. These configurations do not apply to any Postman services.
Learn more about using a custom proxy

- [x] Use the system proxy
  - [ ] Respect HTTP_PROXY, HTTPS_PROXY, and NO_PROXY environment variables.
- [x] Add a custom proxy configuration

Proxy Type	✓ HTTP ✓ HTTPS
Proxy Server	127.0.0.1 : 8080
Proxy Auth	OFF
Username	Username
Password	Password
Proxy Bypass	Enter comma separated hosts to bypass proxy settings. Example: 127.0.0.1, localhost, *.example.com

Figure 4-28: Postman's proxy settings configured to interact with Burp Suite

Try sending a request using Postman; if it is intercepted by Burp Suite, you've properly configured everything. Now you can leave the proxy on and toggle Burp Suite's "turn Intercept on" function when you want to capture requests and responses.

Supplemental Tools

This section is meant to provide additional options and to aid those who are limited by the features available in Burp Suite CE. The following tools are excellent at what they do, open source, and free. In particular, the API scanning tools covered here serve several purposes when you're actively testing your target. Tools such as Nikto and OWASP ZAP can help you actively discover API endpoints, security misconfigurations, and interesting paths, and they provide some surface-level testing of an API. In other words, they are useful when you start actively engaging with a target, whereas tools such as Wfuzz and Arjun will be more useful once you've discovered an API and want to narrow the focus of your testing. Use these tools to actively test APIs to discover unique paths, parameters, files, and functionality. Each of these tools has its own unique focus and purpose that will supplement functionality lacking in the free Burp Suite Community Edition.

Performing Reconnaissance with OWASP Amass

OWASP Amass is an open-source information-gathering tool that can be used for passive and active reconnaissance. This tool was created as a part of the OWASP Amass project, led by Jeff Foley. We will be using Amass to discover the attack surface of our target organizations. With as little as a target's domain name, you can use Amass to scan through many internet sources for your target's associated domains and subdomains to get a list of potential target URLs and APIs.

If OWASP Amass is not installed, use the following command:

```
$ sudo apt-get install amass
```

Amass is pretty effective without much setup. However, you can make it into an information collection powerhouse by setting it up with API keys from various sources. I recommend at least setting up accounts with GitHub, Twitter, and Censys. Once you've set up these accounts, you can generate API keys for these services and plug them into Amass by adding them to Amass's configuration file, *config.ini*. The Amass GitHub repository has a template *config.ini* file that you can use at *https://github.com/OWASP/Amass/blob/master/examples/config.ini*.

On Kali, Amass will attempt to automatically find the *config.ini* file at the following location:

```
$ HOME/.config/amass/config.ini
```

To download the content of the sample *config.ini* file and save it to the default Amass config file location, run the following command from the terminal:

```
$ mkdir $HOME/.config/amass
$ curl https://raw.githubusercontent.com/OWASP/Amass/master/examples/config.ini >$HOME/.config/
amass/config.ini
```

Once you have that file downloaded, you can edit it and add the API keys you would like to include. It should look something like this:

```
# https://umbrella.cisco.com (Paid-Enterprise)
# The apikey must be an API access token created through the Investigate management UI
#[data_sources.Umbrella]
#apikey =

#https://urlscan.io (Free)
#URLScan can be used without an API key
#apikey =

# https://virustotal.com (Free)
#[data_sources.URLScan]
#apikey =
```

As you can see, you can remove the comment (#) and simply paste in the API key for whichever service you would like to use. The *config.ini* file even indicates which keys are free. You can find a list of the sources with APIs you can use to enhance Amass at *https://github.com/OWASP/Amass*. Although it will be a little time-consuming, I recommend taking advantage of at least all the free sources listed under APIs.

Discovering API Endpoints with Kiterunner

Kiterunner (*https://github.com/assetnote/kiterunner*) is a content discovery tool designed specifically for finding API resources. Kiterunner is built with Go, and while it can scan at a speed of 30,000 requests per second, it takes into account the fact that load balancers and web application firewalls will likely enforce rate limiting.

When it comes to APIs, Kiterunner's search techniques outperform other content discovery tools such as dirbuster, dirb, Gobuster, and dirsearch because this tool was built with API awareness. Its wordlists, request methods, parameters, headers, and path structures are all focused on finding API endpoints and resources. Of note, the tool includes data from 67,500 Swagger files. Kiterunner has also been designed to detect the signature of different APIs, including Django, Express, FastAPI, Flask, Nginx, Spring, and Tomcat (just to name a few).

One of the tool's most useful capabilities, which we'll leverage in Chapter 6, is the request replay feature. If Kiterunner detects endpoints when scanning, it will display this result on the command line. You can then dive deeper into the result by exploring the exact request that triggered the result.

To install Kiterunner, run the following commands:

```
$ git clone https://github.com/assetnote/kiterunner.git
$ cd kiterunner
$ make build
$ sudo ln -s $(pwd)/dist/kr /usr/local/bin/kr
```

You should then be able to use Kiterunner from the command line by entering the following:

```
$ kr
kite is a context based webscanner that uses common api paths for content
discovery of an applications api paths.

Usage:
  kite [command]

Available Commands:
  brute       brute one or multiple hosts with a provided wordlist
  help        help about any command
  kb          manipulate the kitebuilder schema
  scan        scan one or multiple hosts with a provided wordlist
  version     version of the binary you're running
  wordlist    look at your cached wordlists and remote wordlists
```

```
Flags:
      --config string   config file (default is $HOME/.kiterunner.yaml)
  -h, --help            help for kite
  -o, --output string   output format. can be json,text,pretty (default
"pretty")
  -q, --quiet           quiet mode. will mute unnecessary pretty text
  -v, --verbose string  level of logging verbosity. can be
error,info,debug,trace (default "info")

Use "kite [command] --help" for more information about a command.
```

You can supply Kiterunner with various wordlists, which it then uses as payloads for a series of requests. These requests will help you discover interesting API endpoints. Kiterunner allows you to use Swagger JSON files, Assetnote's *.kites* files, and *.txt* wordlists. Currently, Assetnote releases its wordlists, which contain search terms collected from its internet-wide scans, on a monthly basis. All of the wordlists are hosted at *https://wordlists.assetnote.io.* Create an API wordlists directory as follows:

```
$ mkdir -p ~/api/wordlists
```

You can then select your desired wordlists and download them to the */api/wordlists* directory:

```
$ curl https://wordlists-cdn.assetnote.io/data/automated/httparchive_apiroutes_2021_06_28.txt >
latest_api_wordlist.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 6651k  100 6651k    0     0  16.1M      0 --:--:-- --:--:-- --:--:-- 16.1M
```

You can replace *httparchive_apiroutes_2021_06_028.txt* with whichever wordlists suit you best. Alternatively, download all the Assetnote wordlists at once:

```
$ wget -r --no-parent -R "index.html*" https://wordlists-cdn.assetnote.io/data/ -nH
```

Be warned that downloading all of the Assetnote wordlists takes up about 2.2GB of space, but storing them is definitely worth it.

Scanning for Vulnerabilities with Nikto

Nikto is a command line web application vulnerability scanner that is quite effective at information gathering. I use Nikto immediately after discovering the existence of a web application, as it can point me toward the application's interesting aspects. Nikto will provide you with information about the target web server, security misconfigurations, and other web application vulnerabilities. Since Nikto is included in Kali, it should not require any special setup.

To scan a domain, use the following command:

```
$ nikto -h https://example.com
```

To see the additional Nikto options, enter **`nikto -Help`** on the command line. A few options you may find useful include `-output` *`filename`* for saving the Nikto results to a specified file and `-maxtime` *`#ofseconds`* to limit how long a Nikto scan will take.

The results from a Nikto scan will include an app's allowed HTTP methods, interesting header information, potential API endpoints, and other directories that could be worth checking out. For additional information about Nikto, review the documentation found at *https://cirt.net/nikto2-docs.*

Scanning for Vulnerabilities with OWASP ZAP

OWASP developed ZAP, an open-source web application scanner, and it's another essential web application security testing tool. OWASP ZAP should be included in Kali, but if it isn't, you can clone it from GitHub at *https://github.com/zaproxy/zaproxy.*

ZAP has two components: automated scan and manual explore. ZAP's *automated scan* performs web crawling, detects vulnerabilities, and tests web application responses by altering request parameters. Automated scan is great for detecting the surface directories of a web application, which includes discovering API endpoints. To run it, enter the target URL into the ZAP interface and click the button to start the attack. Once the scan has run its course, you'll receive a list of alerts that are categorized by the severity of the finding. The issue with ZAP's automated scan is that it can be riddled with false positives, so it is important to examine and validate the alerts. The testing is also limited to the surface of a web application. Unless there are unintentionally exposed directories, ZAP will not be able to infiltrate beyond authentication requirements. This is where the ZAP manual explore option comes in handy.

ZAP *manual explore* is especially useful for exploring beyond the surface of the web application. Also known as the ZAP Heads Up Display (ZAP HUD), manual explore proxies your web browser's traffic through ZAP while you browse. To launch it, enter the URL to explore and open a browser of your choice. When the browser launches, it will appear that you are browsing the site as you normally would; however, ZAP alerts and functions will overlay the web page. This allows you to have much more control over when to start crawling, when to run active scans, and when to turn on "attack mode." For example, you can go through the user account creation process and authentication/authorization process with the ZAP scanner running to automatically detect flaws in these processes. Any vulnerabilities you detect will pop up like gaming achievements. We will be using ZAP HUD to discover APIs.

Fuzzing with Wfuzz

Wfuzz is an open-source Python-based web application fuzzing framework. Wfuzz should come with the latest version of Kali, but you can install it from GitHub at *https://github.com/xmendez/wfuzz.*

You can use Wfuzz to inject a payload within an HTTP request by replacing occurrences of the word *FUZZ* with words from a wordlist; Wfuzz will then rapidly perform many requests (around 900 requests per minute) with the specified payload. Since so much of the success of fuzzing depends on the use of a good wordlist, we'll spend a decent amount of time discussing wordlists in Chapter 6.

Here's the basic request format of Wfuzz:

```
$ wfuzz options -z payload,params url
```

To run Wfuzz, use the following command:

```
$ wfuzz -z file,/usr/share/wordlists/list.txt http://targetname.com/FUZZ
```

This command replaces *FUZZ* in the URL *http://targetname.com/FUZZ* with words from */usr/share/wordlists/list.txt*. The `-z` option specifies a type of payload followed by the actual payload. In this example, we specified that the payload is a file and then provided the wordlist's file path. We could also use `-z` with `list` or `range`. Using the `list` option means that we will specify the payload in the request, whereas `range` refers to a range of numbers. For example, you can use the `list` option to test an endpoint for a list of HTTP verbs:

```
$ wfuzz -X POST -z list,admin-dashboard-docs-api-test http://targetname.com/FUZZ
```

The `-X` option specifies the HTTP request method. In the previous example, Wfuzz will perform a POST request with the wordlist used as the path in place of the *FUZZ* placeholder.

You can use the `range` option to easily scan a series of numbers:

```
$ wfuzz -z range,500-1000 http://targetname.com/account?user_id=FUZZ
```

This will automatically fuzz all numbers from 500 to 1000. This will come in handy when we test for BOLA vulnerabilities.

To specify multiple attack positions, you can list off several `-z` flags and then number the corresponding `FUZZ` placeholders, such as `FUZZ`, `FUZ1`, `FUZ2`, `FUZ3`, and so on, like so:

```
$ wfuzz -z list,A-B-C -z range,1-3 http://targetname.com/FUZZ/user_id=FUZZ2
```

Running Wfuzz against a target can generate a ton of results, which can make it difficult to find anything interesting. Therefore, you should familiarize yourself with the Wfuzz filter options. The following filters display only certain results:

`--sc`	Only shows responses with specific HTTP response codes
`--sl`	Only shows responses with a certain number of lines
`--sw`	Only shows responses with a certain number of words
`--sh`	Only shows responses with a certain number of characters

In the following example, Wfuzz will scan the target and only show results that include a status code of 200:

```
$ wfuzz -z file,/usr/share/wordlists/list.txt -sc 200 http://targetname.com/FUZZ
```

The following filters hide certain results:

`--hc`	Hides responses with specific HTTP status codes
`--hl`	Hides responses with a specified number of lines
`--hw`	Hides responses with a specified number of words
`--hh`	Hides responses with specified number of characters

In the following example, Wfuzz will scan the target and hide all results that have a status code of 404 and hide results that have 950 characters:

```
$ wfuzz -z file,/usr/share/wordlists/list.txt -sc 404 -sh 950 http://targetname.com/FUZZ
```

Wfuzz is a powerful multipurpose fuzzing tool you can use to thoroughly test endpoints and find their weaknesses. For more information about Wfuzz, check out the documentation at *https://wfuzz.readthedocs.io/en/latest.*

Discovering HTTP Parameters with Arjun

Arjun is another open source Python-based API fuzzer developed specifically to discover web application parameters. We will use Arjun to discover basic API functionality, find hidden parameters, and test API endpoints. You can use it as a great first scan for an API endpoint during black box testing or as an easy way to see how well an API's documented parameters match up with the scan's findings.

Arjun comes configured with a wordlist containing nearly 26,000 parameters, and unlike Wfuzz, it does some of the filtering for you using its preconfigured anomaly detection. To set up Arjun, first clone it from GitHub (you'll need a GitHub account to do this):

```
$ cd /opt/
$ sudo git clone https://github.com/s0med3v/Arjun.git
```

Arjun works by first performing a standard request to the target API endpoint. If the target responds with HTML forms, Arjun will add the form names to the parameter list during its scan. Arjun then sends a request with parameters it expects to return responses for nonexistent resources. This is done to note the behavior of a failed parameter request. Arjun then kicks off 25 requests containing the payload of nearly 26,000 parameters, compares the API endpoint's responses, and begins additional scans of the anomalies.

To run Arjun, use the following command:

```
$ python3 /opt/Arjun/arjun.py -u http://target_address.com
```

If you would like to have the output results in a certain format, use the `-o` option with your desired file type:

```
$ python3 /opt/Arjun/arjun.py -u http://target_address.com -o arjun_results.json
```

If you come across a target with rate limiting, Arjun may trigger the rate limit and cause a security control to block you. Arjun even has built-in suggestions for when a target does not cooperate. Arjun may prompt you with an error message such as "Target is unable to process requests, try --stable switch." If this happens, simply add the `--stable` flag. Here's an example:

```
$ python3 /opt/Arjun/arjun.py -u http://target_address.com -o arjun_results.json --stable
```

Finally, Arjun can scan multiple targets at once. Use the `-i` flag to specify a list of target URLs. If you've been proxying traffic with Burp Suite, you can select all URLs within the sitemap, use the Copy Selected URLs option, and paste that list to a text file. Then run Arjun against all Burp Suite targets simultaneously, like this:

```
$ python3 /opt/Arjun/arjun.py -i burp_targets.txt
```

Summary

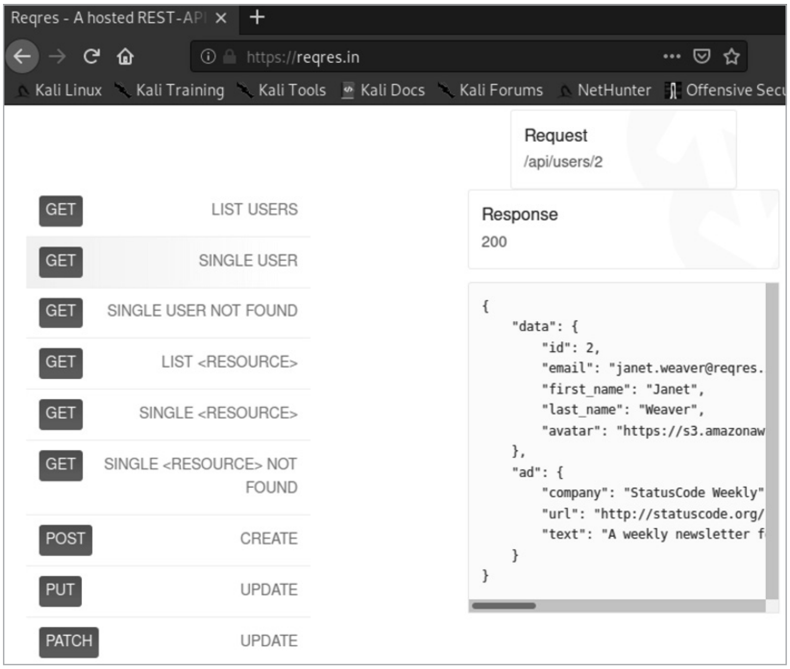
In this chapter, you set up the various tools we'll use to hack APIs throughout this book. Additionally, we spent some time digging into feature-rich applications such as DevTools, Burp Suite, and Postman. Being comfortable with the API hacking toolbox will help you know when to use which tool and when to pivot.

Lab #1: Enumerating the User Accounts in a REST API

Welcome to your first lab.

In this lab, our goal is simple: find the total number of user accounts in *reqres.in*, a REST API designed for testing, using the tools discussed in this chapter. You could easily figure this out by guessing the total number of accounts and then checking for that number, but we will discover the answer much more quickly using the power of Postman and Burp Suite. When testing actual targets, you could use this process to discover whether there was a basic BOLA vulnerability present.

First, navigate to *https://reqres.in* to see if API documentation is available. On the landing page, we find the equivalent of API documentation and can see a sample request that consists of making a request to the */api/users/2* endpoint (see Figure 4-29).

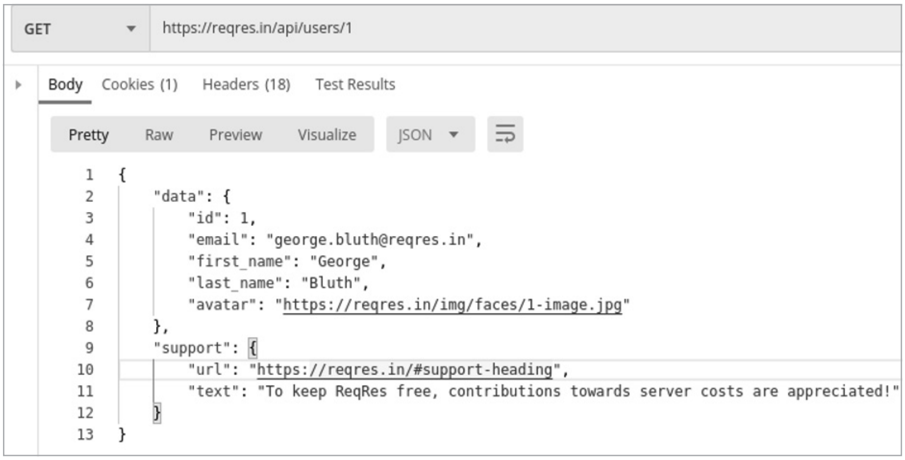


```
{
    "data": {
        "id": 2,
        "email": "janet.weaver@reqres.
        "first_name": "Janet",
        "last_name": "Weaver",
        "avatar": "https://s3.amazonaw
    },
    "ad": {
        "company": "StatusCode Weekly"
        "url": "http://statuscode.org/
        "text": "A weekly newsletter f
    }
}
```

Figure 4-29: API documentation found at https://reqres.in *with instructions for requesting user* `id:2`

You'll notice a List Users endpoint; we'll ignore this for the purposes of the lab, as it won't help you learn the intended concepts. Instead, we'll be using the Single User endpoint because it will help you build the skills needed to discover vulnerabilities like BOLA and BFLA. The suggested API request for Single User is meant to provide the consumer with the requested user's account information by sending a GET request to */api/users/*. We can easily assume that user accounts are organized in the *user* directory by their `id` number.

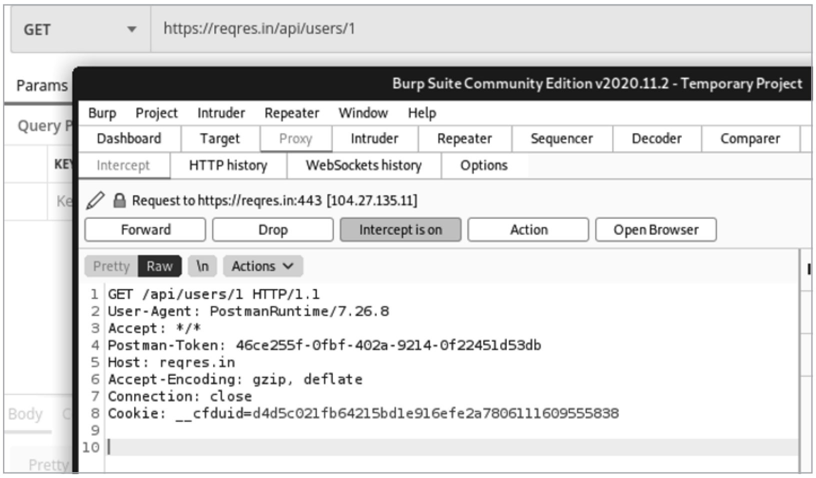
Let's test this theory by attempting to send a request to a user with a different ID number. Since we'll be interacting with an API, let's set up the API request using Postman. Set the method to GET and add the URL *http://reqres.in/api/users/1*. Click **Send** and make sure you get a response. If you requested the user with an ID of 1, the response should reveal the user information for George Bluth, as seen in Figure 4-30.



```
{
    "data": {
        "id": 1,
        "email": "george.bluth@reqres.in",
        "first_name": "George",
        "last_name": "Bluth",
        "avatar": "https://reqres.in/img/faces/1-image.jpg"
    },
    "support": {
        "url": "https://reqres.in/#support-heading",
        "text": "To keep ReqRes free, contributions towards server costs are appreciated!"
    }
}
```

Figure 4-30: A standard API request made using Postman to retrieve user 1 from the https://reqres.in *database*

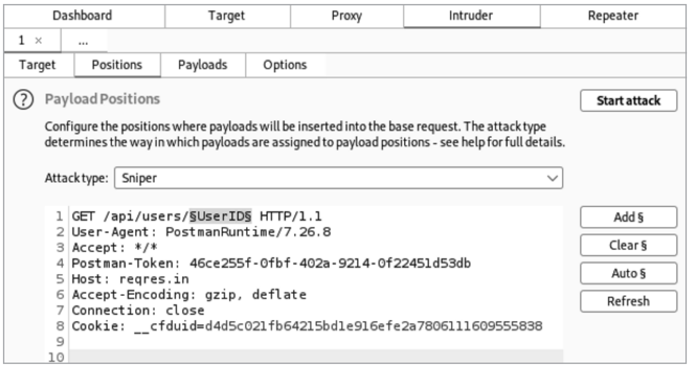
To efficiently retrieve the data of all users by following this method, we'll use Burp's Intruder. Proxy the traffic from the *reqres.in* endpoint over to Burp Suite and submit the same request in Postman. Migrate over to Burp Suite, where you should see the intercepted traffic in Burp Suite's Proxy tab (see Figure 4-31).



```
GET /api/users/1 HTTP/1.1
User-Agent: PostmanRuntime/7.26.8
Accept: */*
Postman-Token: 46ce255f-0fbf-402a-9214-0f22451d53db
Host: reqres.in
Accept-Encoding: gzip, deflate
Connection: close
Cookie: __cfduid=d4d5c021fb64215bd1e916efe2a7806111609555838
```

Figure 4-31: The intercepted request made using Postman to retrieve user 1

Use the shortcut CTRL-I or right-click the intercepted request and select **Send to Intruder**. Select the **Intruder ▸ Positions** tab to select the payload positions. First, select **Clear §** to remove the automatic payload positioning. Then select the number at the end of the URL and click the button labeled **Add §** (see Figure 4-32).



Payload Positions

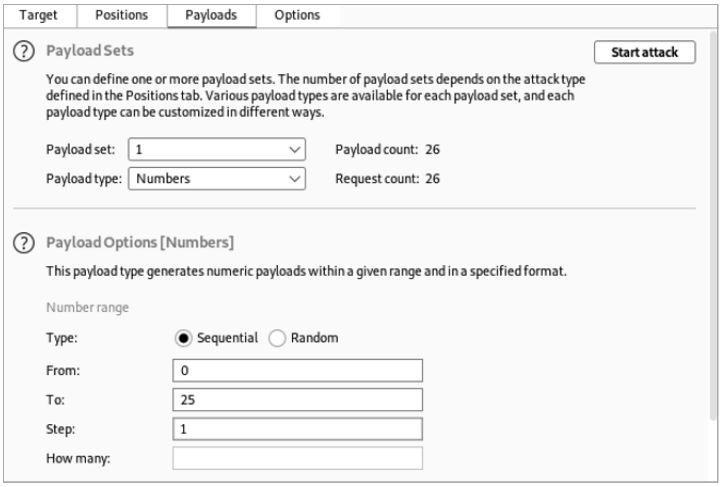
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
GET /api/users/§UserID§ HTTP/1.1
User-Agent: PostmanRuntime/7.26.8
Accept: */*
Postman-Token: 46ce255f-0fbf-402a-9214-0f22451d53db
Host: reqres.in
Accept-Encoding: gzip, deflate
Connection: close
Cookie: __cfduid=d4d5c021fb64215bd1e916efe2a7806111609555838
```

Figure 4-32: Burp Suite's Intruder configured with the attack position set around the UserID *portion of the path*

Once you've selected the attack position, select the **Payloads** tab (see Figure 4-33). Since our goal is to find out how many user accounts exist, we want to replace the user ID with a series of numbers. Change the payload type to **Numbers**. Update the range of numbers to test from 0 to 25, stepping by 1. The Step option indicates to Burp how many numbers to increase with each payload. By selecting 1, we are letting Burp do the heavy lifting of creating all the payloads on the fly. This will help us discover all the users with an ID between 0 and 25. With these settings, Burp will send a total of 26 requests, each one with a number from 0 to 25.



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 26
Payload type: Numbers Request count: 26

Payload Options [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range
Type: (● Sequential) ○ Random
From: 0
To: 25
Step: 1
How many:

Figure 4-33: Intruder's Payloads tab with the payload type set to numbers

Finally, click **Start Attack** to send the 26 requests to *reqres.in*. Analyzing the results should give you a clear indication of all the live users. The API provider responds with a status 200 for user accounts between 1 and 12 and a status of 404 for the subsequent requests. Judging by the results, we can conclude that this API has a total of 12 valid user accounts.

Of course, this was just practice. The values you replace in a future API hacking engagement could be user ID numbers, but they could just as easily be bank account numbers, phone numbers, company names, or email addresses. This lab has prepared you to take on the world of basic BOLA vulnerabilities; we will expand on this knowledge in Chapter 10.

As a further exercise, try performing this same scan using Wfuzz.

5
SETTING UP VULNERABLE API TARGETS

In this chapter, you'll build your own API target lab to attack in subsequent chapters. By targeting a system you control, you'll be able to safely practice your techniques and see their impacts from both the offensive and defensive perspectives. You'll also be able to make mistakes and experiment with exploits you may not yet be comfortable with using in real engagements.

You'll be targeting these machines throughout the lab sections in this book to find out how tools work, discover API weaknesses, learn to fuzz inputs, and exploit all your findings. The lab will have vulnerabilities well beyond what is covered in this book, so I encourage you to seek them out and develop new skills through experimentation.

This chapter walks you through setting up prerequisites in a Linux host, installing Docker, downloading and launching the three vulnerable systems that will be used as our targets, and finding additional resources for API hacking targets.

NOTE *This lab contains deliberately vulnerable systems. These could attract attackers and introduce new risks to your home or work networks. Do not connect these machines to the rest of your network; make sure the hacking lab is isolated and protected. In general, be aware of where you host a network of vulnerable machines.*

Creating a Linux Host

You'll need a host system to be able to run three vulnerable applications. For the sake of simplicity, I recommend keeping the vulnerable applications on different host systems. When they are hosted together, you could run into conflicts in the resources the applications use, and an attack on one vulnerable web app could affect the others. It is easier to be able to have each vulnerable app on its own host system.

I recommend using a recent Ubuntu image hosted either on a hypervisor (such as VMware, Hyper-V, or VirtualBox) or in the cloud (such as AWS, Azure, or Google Cloud). The basics of setting up host systems and networking them together is beyond the scope of this book and is widely covered elsewhere. You can find many excellent free guides out there for setting up the basics of a home or cloud hacking lab. Here are a few I recommend:

Cybrary, "Tutorial: Setting Up a Virtual Pentesting Lab at Home," *https://www.cybrary.it/blog/0p3n/tutorial-for-setting-up-a-virtual-penetration-testing-lab-at-your-home*

Black Hills Information Security, "Webcast: How to Build a Home Lab," *https://www.blackhillsinfosec.com/webcast-how-to-build-a-home-lab*

Null Byte, "How to Create a Virtual Hacking Lab," *https://null-byte.wonderhowto.com/how-to/hack-like-pro-create-virtual-hacking-lab-0157333*

Hacking Articles, "Web Application Pentest Lab Setup on AWS," *https://www.hackingarticles.in/web-application-pentest-lab-setup-on-aws*

Use these guides to set up your Ubuntu machine.

Installing Docker and Docker Compose

Once you've configured your host operating system, you can use Docker to host the vulnerable applications in the form of containers. Docker and Docker Compose will make it incredibly easy to download the vulnerable apps and launch them within a few minutes.

Follow the official instructions at *https://docs.docker.com/engine/install/ubuntu* to install Docker on your Linux host. You'll know that Docker Engine is installed correctly when you can run the hello-world image:

```
$ sudo docker run hello-world
```

If you can run the hello-world container, you have successfully set up Docker. Congrats! Otherwise, you can troubleshoot using the official Docker instructions.

Docker Compose is a tool that will enable you to run multiple containers from a YAML file. Depending on your hacking lab setup, Docker Compose could allow you to launch your vulnerable systems with the simple command `docker-compose up`. The official documentation for installing Docker Compose can be found at *https://docs.docker.com/compose/install.*

Installing Vulnerable Applications

I have selected these vulnerable applications to run in the lab: OWASP crAPI, OWASP Juice Shop, OWASP DevSlop's Pixi, and Damn Vulnerable GraphQL. These apps will help you develop essential API hacking skills such as discovering APIs, fuzzing, configuring parameters, testing authentication, discovering OWASP API Security Top 10 vulnerabilities, and attacking discovered vulnerabilities. This section describes how to set up these applications.

The completely ridiculous API (crAPI)

The completely ridiculous API, shown in Figure 5-1, is the vulnerable API developed and released by the OWASP API Security Project. As noted in the acknowledgments of this book, this project was led by Inon Shkedy, Erez Yalon, and Paolo Silva. The crAPI vulnerable API was designed to demonstrate the most critical API vulnerabilities. We will focus on hacking crAPI during most of our labs.

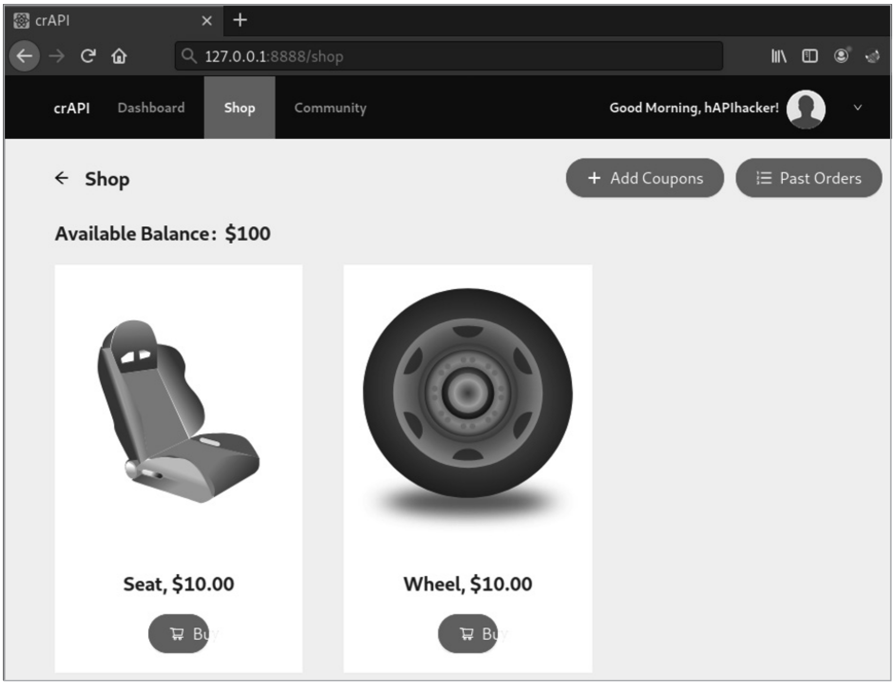


Figure 5-1: The crAPI shop

The crAPI application contains a modern web application, an API, and a Mail Hog email server. In this application, you can shop for vehicle parts, use the community chat feature, and link a vehicle to find local repair shops. The crAPI app was built with realistic implementations of the OWASP API Security Top 10 vulnerabilities. You will learn quite a bit from this one.

OWASP DevSlop's Pixi

Pixi is a MongoDB, Express.js, Angular, Node (MEAN) stack web application that was designed with deliberately vulnerable APIs (see Figure 5-2). It was created at OWASP DevSlop, an OWASP incubator project that highlights DevOps-related mistakes, by Nicole Becher, Nancy Gariché, Mordecai Kraushar, and Tanya Janca.

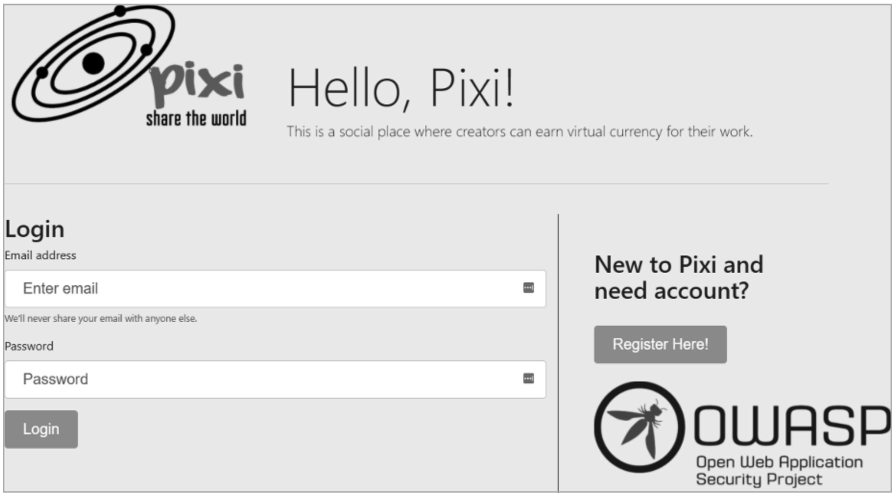

pixi
share the world
Hello, Pixi!
This is a social place where creators can earn virtual currency for their work.
Login
Email address
Enter email
We'll never share your email with anyone else.
Password
Password
Login
New to Pixi and need account?
Register Here!
OWASP
Open Web Application Security Project

Figure 5-2: The Pixi landing page

You can think of the Pixi application as a social media platform with a virtual payment system. As an attacker, you'll find Pixi's user information, administrative functionality, and payment system especially interesting.

Another great feature of Pixi is that it is very easy to get up and running. Run the following commands from an Ubuntu terminal:

```
$ git clone https://github.com/DevSlop/Pixi.git
$ cd Pixi
$ sudo docker-compose up
```

Then use a browser and visit *http://localhost:8000* to see the landing page. If Docker and Docker Compose have been set up, as described previously in this chapter, launching Pixi should really be as easy as that.

OWASP Juice Shop

OWASP Juice Shop, shown in Figure 5-3, is an OWASP flagship project created by Björn Kimminich. It's designed to include vulnerabilities from both the OWASP Top 10 and OWASP API Security Top 10. One awesome feature found in Juice Shop is that it tracks your hacking progress and includes a hidden scoreboard. Juice Shop was built using Node.js, Express, and Angular. It is a JavaScript application powered by REST APIs.

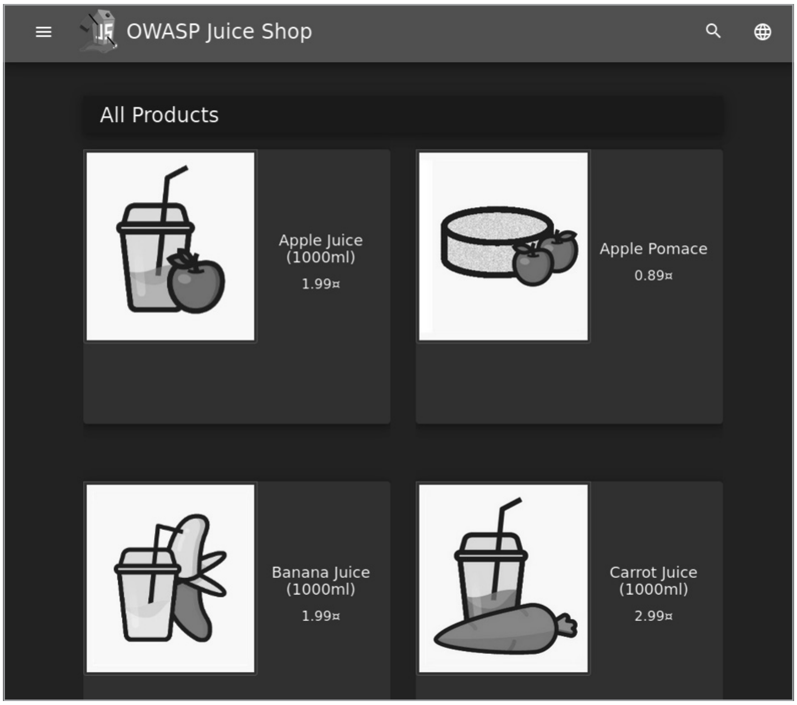


Figure 5-3: The OWASP Juice Shop

Of all the applications we'll install, Juice Shop is currently the most supported, with over 70 contributors. To download and launch Juice Shop, run the following commands:

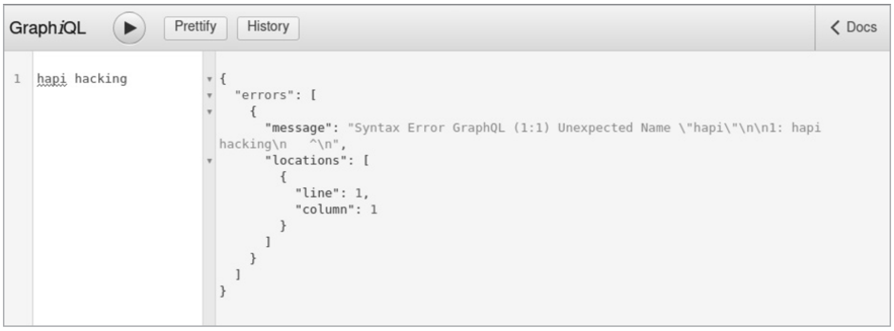
```
$ docker pull bkimminich/juice-shop
$ docker run --rm -p 80:3000 bkimminich/juice-shop
```

Juice Shop and Damn Vulnerable GraphQL Application (DVGA) both run over port 3000 by default. To avoid conflict, the `-p 80:3000` argument in the `docker-run` command sets Juice Shop up to run over port 80 instead.

To access Juice Shop, browse to *http://localhost.* (On macOS and Windows, browse to *http://192.168.99.100* if you are using Docker Machine instead of the native Docker installation.)

Damn Vulnerable GraphQL Application

DVGA is a deliberately vulnerable GraphQL application developed by Dolev Farhi and Connor McKinnon. I'm including DVGA in this lab because of GraphQL's increasing popularity and adoption by organizations such as Facebook, Netflix, AWS, and IBM. Additionally, you may be surprised by how often a GraphQL integrated development environment (IDE) is exposed for all to use. GraphiQL is one of the more popular GraphQL IDEs you will come across. Understanding how to take advantage of the GraphiQL IDE will prepare you to interact with other GraphQL APIs with or without a friendly user interface (see Figure 5-4).



```
hapi hacking
```

```
{
  "errors": [
    {
      "message": "Syntax Error GraphQL (1:1) Unexpected Name \"hapi\"\n\n1: hapi
hacking\n   ^\n",
      "locations": [
        {
          "line": 1,
          "column": 1
        }
      ]
    }
  ]
}
```

Figure 5-4: The GraphiQL IDE web page hosted on port 5000

To download and launch DVGA, run the following commands from your Ubuntu host terminal:

```
$ sudo docker pull dolevf/dvga
$ sudo docker run -t -p 5000:5000 -e WEB_HOST=0.0.0.0 dolevf/dvga
```

To access it, use a browser and visit *http://localhost:5000*.

Adding Other Vulnerable Apps

If you are interested in an additional challenge, you can add other machines to your API hacking lab. GitHub is a great source of deliberately vulnerable APIs to bolster your lab. Table 5-1 lists a few more systems with vulnerable APIs you can easily clone from GitHub.

Table 5-1: Additional Systems with Vulnerable APIs

Name	Contributor	GitHub URL
VAmPI	Erev0s	*https://github.com/erev0s/VAmPI*
DVWS-node	Snoopysecurity	*https://github.com/snoopysecurity/dvws-node*
DamnVulnerable MicroServices	ne0z	*https://github.com/ne0z/DamnVulnerableMicroServices*
Node-API-goat	Layro01	*https://github.com/layro01/node-api-goat*
Vulnerable GraphQL API	AidanNoll	*https://github.com/CarveSystems/vulnerable-graphql-api*
Generic-University	InsiderPhD	*https://github.com/InsiderPhD/Generic-University*
vulnapi	tkisason	*https://github.com/tkisason/vulnapi*

Hacking APIs on TryHackMe and HackTheBox

TryHackMe (*https://tryhackme.com*) and HackTheBox (*https://www.hackthebox.com*) are web platforms that allow you to hack vulnerable machines, participate in capture-the-flag (CTF) competitions, solve hacking challenges, and climb hacking leaderboards. TryHackMe has some free content and much more content for a monthly subscription fee. You can deploy its prebuilt hacking machines over a web browser and attack them. It includes several great machines with vulnerable APIs:

- Bookstore (free)
- Carpe Diem 1 (free)
- ZTH: Obscure Web Vulns (paid)
- ZTH: Web2 (paid)
- GraphQL (paid)

These vulnerable TryHackMe machines cover many of the basic approaches to hacking REST APIs, GraphQL APIs, and common API authentication mechanisms. If you're new to hacking, TryHackMe has made deploying an attacking machine as simple as clicking Start Attack Box. Within a few minutes, you'll have a browser-based attacking machine with many of the tools we will be using throughout this book.

HackTheBox (HTB) also has free content and a subscription model but assumes you already have basic hacking skills. For example, HTB does not currently provide users with attacking machine instances, so it requires you to come prepared with your own attacking machine. In order to use HTB at all, you need to be able to take on its challenge and hack its invitation code process to gain entry.

The primary difference between the HTB free tier and its paid tier is access to vulnerable machines. With free access, you'll have access to the 20 most recent vulnerable machines, which may include an API-related system. However, if you want access to HTB's library of vulnerable machines with API vulnerabilities, you will need to pay for a VIP membership that lets you access its retired machines.

The retired machines listed in Table 5-2 all include aspects of API hacking.

Table 5-2: Retired Machines with API Hacking Components

Craft	Postman	Smasher2
JSON	Node	Help
PlayerTwo	Luke	Playing with Dirty Socks

HTB provides one of the best ways to improve your hacking skills and expand your hacking lab experience beyond your own firewall. Outside of the HTB machines, challenges such as Fuzzy can help you improve critical API hacking skills.

Web platforms like TryHackMe and HackTheBox are great supplements to your hacking lab and will help boost your API hacking abilities. When you're not out hacking in the real world, you should keep your skills sharp with CTF competitions like these.

Summary

In this chapter, I guided you through setting up your own set of vulnerable applications that you can host in a home lab. As you learn new skills, the applications in this lab will serve as a place to practice finding and exploiting API vulnerabilities. With these vulnerable apps running in your home lab, you will be able to follow along with the tools and techniques used in the following chapters and lab exercises. I encourage you to go beyond my recommendations and learn new things on your own by expanding or adventuring beyond this API hacking lab.

Lab #2: Finding Your Vulnerable APIs

Let's get your fingers on the keyboard. In this lab, we'll use some basic Kali tools to discover and interact with the vulnerable APIs you just set up. We'll search for the Juice Shop lab application on our local network using Netdiscover, Nmap, Nikto, and Burp Suite.

NOTE *This lab assumes you've hosted the vulnerable applications on your local network or on a hypervisor. If you've set up this lab in the cloud, you won't need to discover the IP address of the host system, as you should have that information.*

Before powering up your lab, I recommend getting a sense of what devices can be found on your network. Use Netdiscover before starting up the vulnerable lab and after you have the lab started:

```
$ sudo netdiscover
Currently scanning: 172.16.129.0/16   |   Screen View: Unique Hosts

 13 Captured ARP Req/Rep packets, from 4 hosts.   Total size: 780

-----------------------------------------------------------------------------
   IP            At MAC Address     Count     Len  MAC Vendor / Hostname
-----------------------------------------------------------------------------
 192.168.195.2   00:50:56:f0:23:20      6     360  VMware, Inc.
 192.168.195.130 00:0c:29:74:7c:5d      4     240  VMware, Inc.
 192.168.195.132 00:0c:29:85:40:c0      2     120  VMware, Inc.
 192.168.195.254 00:50:56:ed:c0:7c      1      60  VMware, Inc.
```

You should see a new IP address appear on the network. Once you've discovered the vulnerable lab IP, you can use CTRL-C to stop Netdiscover.

Now that you have the IP address of the vulnerable host, find out what services and ports are in use on that virtual device with a simple Nmap command:

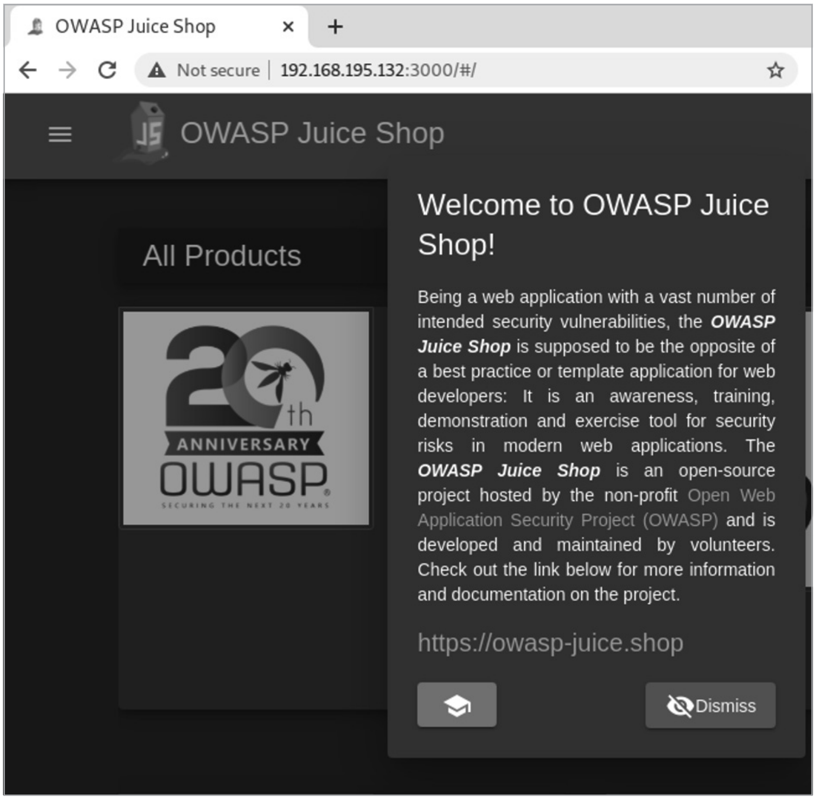
```
$ nmap 192.168.195.132
Nmap scan report for 192.168.195.132
Host is up (0.00046s latency).
Not shown: 999 closed ports
PORT         STATE         SERVICE
3000/tcp     open          ppp

Nmap done: 1 IP address (1 host up) scanned in 0.14 seconds
```

We can see that the targeted IP address has only port 3000 open (which matches up with what we'd expect based on our initial setup of Juice Shop). To find out more information about the target, we can add the -sC and -sV flags to our scan to run default Nmap scripts and to perform service enumeration:

```
$ nmap -sC -sV 192.168.195.132
Nmap scan report for 192.168.195.132
Host is up (0.00047s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE VERSION
3000/tcp open  ppp?
| fingerprint-strings:
|   DNSStatusRequestTCP, DNSVersionBindReqTCP, Help, NCP, RPCCheck, RTSPRequest:
|     HTTP/1.1 400 Bad Request
|     Connection: close
|   GetRequest:
     HTTP/1.1 200 OK
--snip--
     Copyright (c) Bjoern Kimminich.
     SPDX-License-Identifier: MIT
     <!doctype html>
     <html lang="en">
     <head>
     <meta charset="utf-8">
     <title>OWASP Juice Shop</title>
```

By running this command, we learn that HTTP is running over port 3000. We've found a web app titled "OWASP Juice Shop." Now we should be able to use a web browser to access Juice Shop by navigating to the URL (see Figure 5-5). In my case, the URL is *http://192.168.195.132:3000.*


OWASP Juice Shop

192.168.195.132:3000/#/

OWASP Juice Shop

All Products

Welcome to OWASP Juice Shop!

Being a web application with a vast number of intended security vulnerabilities, the ***OWASP Juice Shop*** is supposed to be the opposite of a best practice or template application for web developers: It is an awareness, training, demonstration and exercise tool for security risks in modern web applications. The ***OWASP Juice Shop*** is an open-source project hosted by the non-profit Open Web Application Security Project (OWASP) and is developed and maintained by volunteers. Check out the link below for more information and documentation on the project.

https://owasp-juice.shop

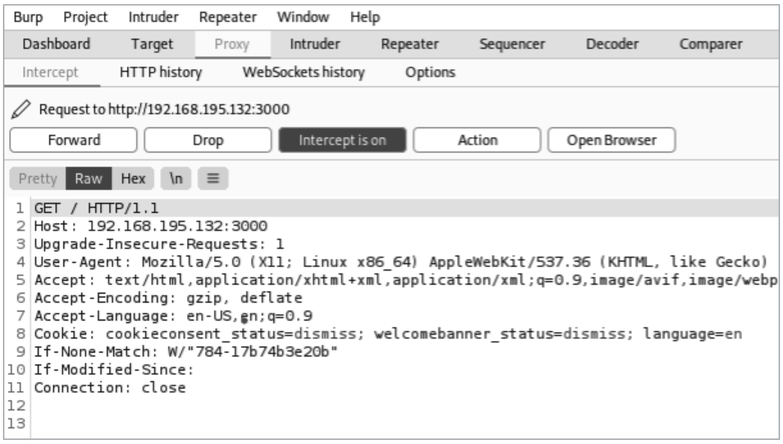
Figure 5-5: OWASP Juice Shop

At this point, you can explore the web application with your web browser, see its various features, and find the fine juices of the Juice Shop. In general, click things and pay attention to the URLs these clicks generate for signs of APIs at work. A typical first step after exploring the web application is to test it for vulnerabilities. Use the following Nikto command to scan the web app in your lab:

```
$ nikto -h http://192.168.195.132:3000
---------------------------------------------------------------------------
+ Target IP:          192.168.195.132
+ Target Hostname:    192.168.195.132
+ Target Port:        3000
---------------------------------------------------------------------------
+ Server: No banner retrieved
+ Retrieved access-control-allow-origin header: *
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect
against some forms of XSS
+ Uncommon header 'feature-policy' found, with contents: payment 'self'
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Entry '/ftp/' in robots.txt returned a non-forbidden or redirect HTTP code (200)
+ "robots.txt" contains 1 entry which should be manually viewed.
```

Nikto highlights some juicy information, such as the *robots.txt* file and a valid entry for FTP. However, nothing here reveals that an API is at work.

Since we know that APIs operate beyond the GUI, it makes sense to begin capturing web traffic by proxying our traffic through Burp Suite. Make sure to set FoxyProxy to your Burp Suite entry and confirm that Burp Suite has the Intercept option switched on (see Figure 5-6). Next, refresh the Juice Shop web page.


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer
Intercept HTTP history WebSockets history Options
Request to http://192.168.195.132:3000
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex \n

```
GET / HTTP/1.1
Host: 192.168.195.132:3000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: cookieconsent_status=dismiss; welcomebanner_status=dismiss; language=en
If-None-Match: W/"784-17b74b3e20b"
If-Modified-Since:
Connection: close
```

Figure 5-6: An intercepted Juice Shop HTTP request

Once you've intercepted a request with Burp Suite, you should see something similar to what's shown in Figure 5-6. However, still no APIs! Next, slowly click **Forward** to send one automatically generated request after another to the web application and notice how the web browser's GUI slowly builds.

Once you start forwarding requests, you should see the following, indicating API endpoints:

```
GET /rest/admin/application-configuration
GET /api/Challenges/?name=Score%20Board
GET /api/Quantitys/
```

Nice! This short lab demonstrated how you can search for a vulnerable machine in your local network environment. We performed some basic usage of the tools we set up in Chapter 4 to help us find one of the vulnerable applications and capture some interesting-looking API requests being sent beyond what we can normally see in the web browser's GUI.