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 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 Paulo 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.

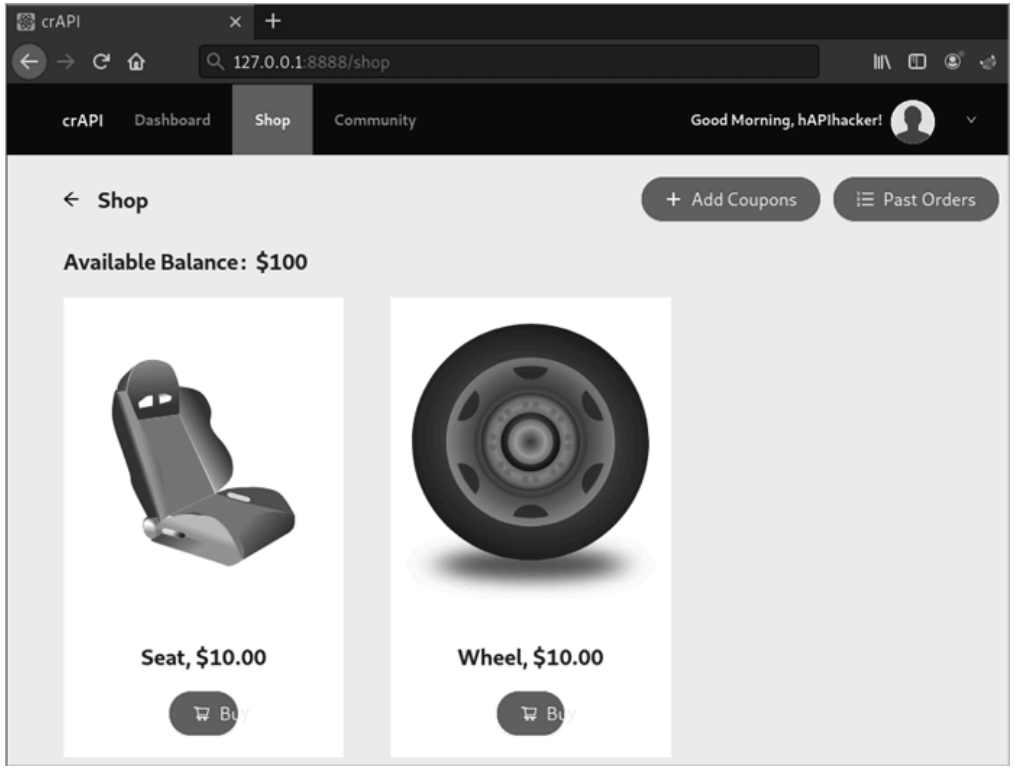

Figure 5-1: The crAPI shop

Download and deploy crAPI (*https://github.com/OWASP/crAPI*) by running the following commands from an Ubuntu terminal:

```
$ curl -o docker-compose.yml https://raw.githubusercontent.com/OWASP/crAPI/main/deploy/docker/docker-compose
$ sudo docker-compose pull
$ sudo docker-compose -f docker-compose.yml --compatibility up -d
```

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.

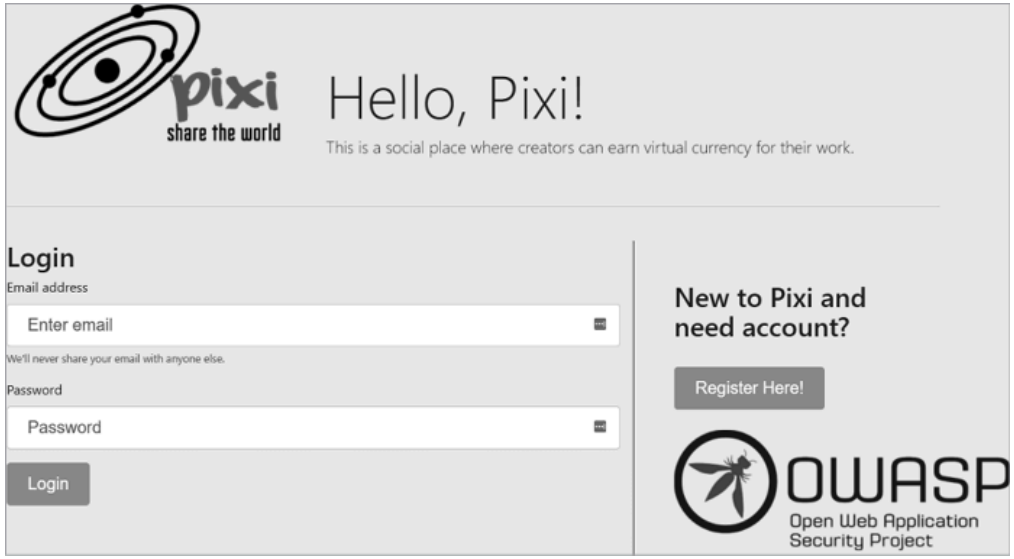

Figure 5-2: The Pixi landing page

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:

```
$ 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.

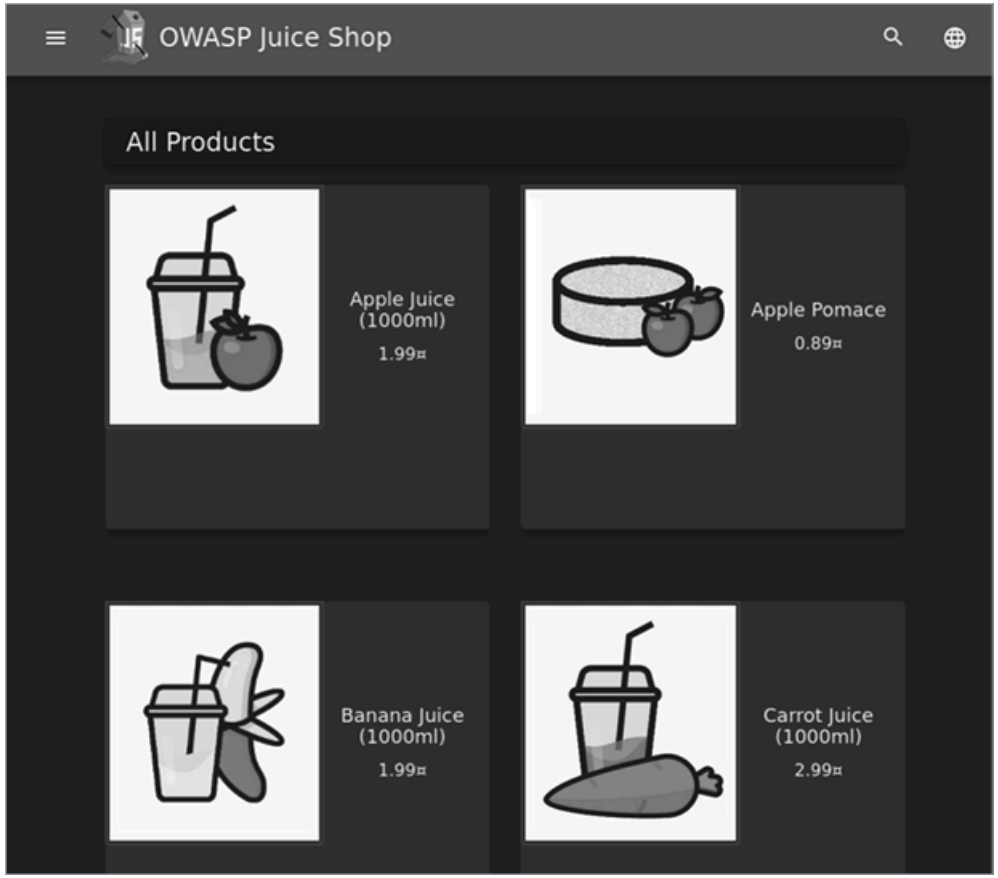

Figure 5-3: The OWASP Juice Shop

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*).

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*.

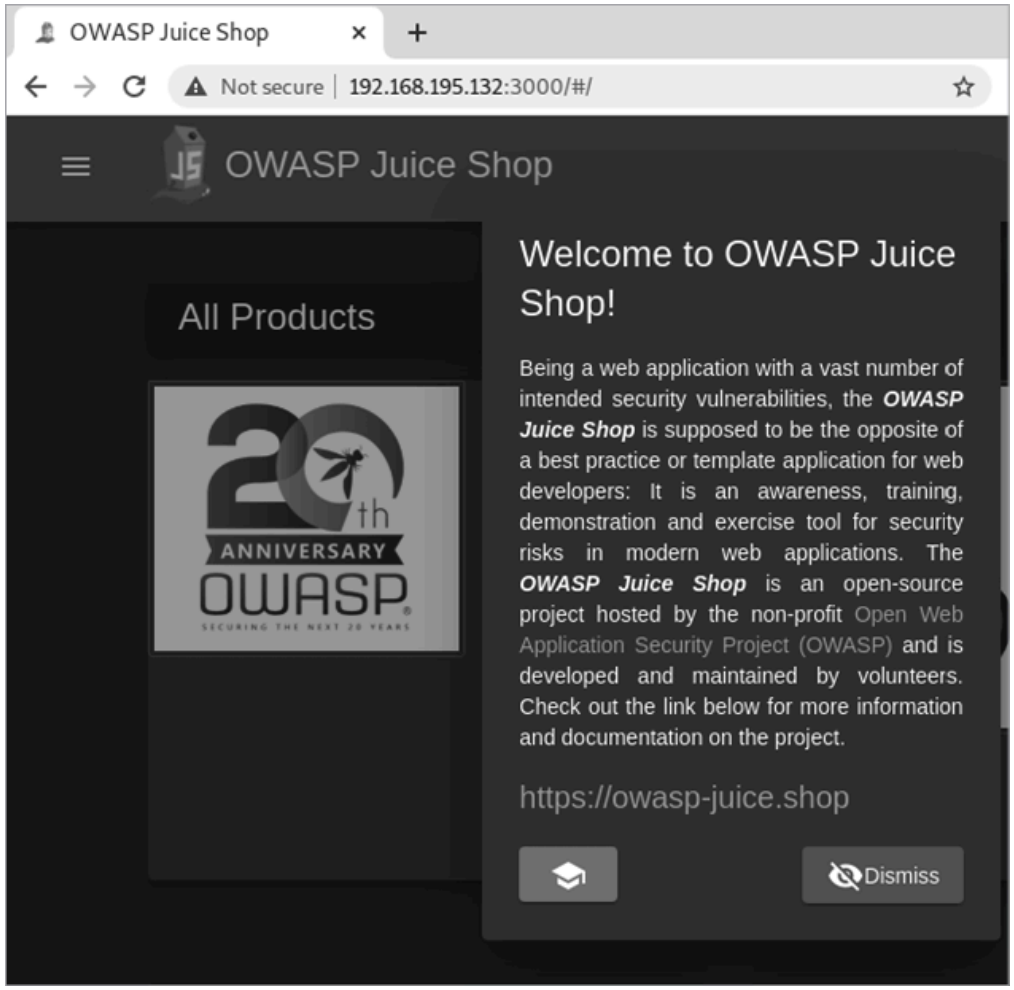

Figure 5-5: OWASP Juice Shop

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
+ 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.

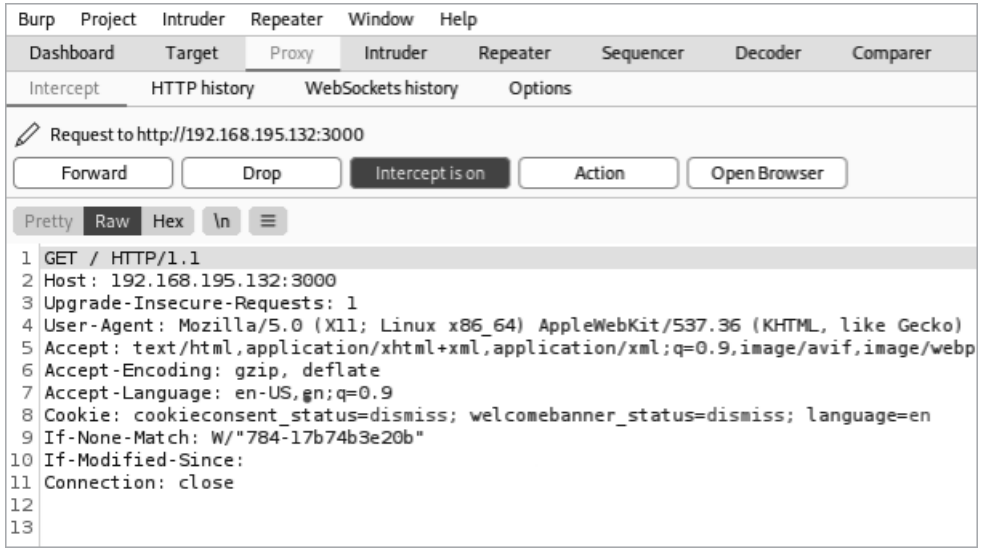

Figure 5-6: An intercepted Juice Shop HTTP request

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.