0
PREPARING FOR YOUR SECURITY TESTS

API security testing does not quite fit into the mold of a general penetration test, nor does it fit into that of a web application penetration test. Due to the size and complexity of many organizations' API attack surfaces, API penetration testing is its own unique service. In this chapter I will discuss the features of APIs that you should include in your test and document prior to your attack. The content in this chapter will help you gauge the amount of activity required for an engagement, ensure that you plan to test all features of the target APIs, and help you avoid trouble.

API penetration testing requires a well-developed *scope*, or an account of the targets and features of what you are allowed to test, that ensures the client and tester have a mutual understanding of the work being done. Scoping an API security testing engagement comes down to a few factors: your methodology, the magnitude of the testing, the target features, any restrictions on testing, your reporting requirements, and whether you plan to conduct remediation testing.

Receiving Authorization

Before you attack APIs, it is supremely important that you receive a signed contract that includes the scope of the engagement and grants you authorization to attack the client's resources within a specific time frame.

For an API penetration test, this contract can take the form of a signed statement of work (SOW) that lists the approved targets, ensuring that you and your client agree on the service they want you to provide. This includes coming to an agreement over which aspects of an API will be tested, determining any exclusions, and setting up an agreed-upon time to perform testing.

Double-check that the person signing the contract is a representative of the target client who is in a position to authorize testing. Also make sure the assets to be tested are owned by the client; otherwise, you will need to rinse and repeat these instructions with the proper owner. Remember to take into consideration the location where the client is hosting their APIs and whether they are truly in a position to authorize testing against both the software and the hardware.

Some organizations can be too restrictive with their scoping documentation. If you have the opportunity to develop the scope, I recommend that, in your own calm words, you kindly explain to your clients that the criminals have no scope or limitations. Real criminals do not consider other projects that are consuming IT resources; they do not avoid the subnet with sensitive production servers or care about hacking at inconvenient times of day. Make an effort to convince your client

of the value of having a less-restrictive engagement and then work with them to document the particulars.

Meet with the client, spell out exactly what is going to happen, and then document it exactly in the contract, reminder emails, or notes. If you stick to the documented agreement for the services requested, you should be operating legally and ethically. However, it is probably worth reducing your risk by consulting with a lawyer or your legal department.

Threat Modeling an API Test

Threat modeling is the process used to map out the threats to an API provider. If you model an API penetration test based on a relevant threat, you'll be able to choose tools and techniques directed at that attack. The best tests of an API will be those that align with actual threats to the API provider.

A *threat actor* is the adversary or attacker of the API. The adversary can be anyone, from a member of the public who stumbles upon the API with little to no knowledge of the application to a customer using the application, a rogue business partner, or an insider who knows quite a bit about the application. To perform a test that provides the most value to the security of the API, it is ideal to map out the probable adversary as well as their hacking techniques.

Your testing method should follow directly from the threat actor's perspective, as this perspective should determine the information you are given about your target. If the threat actor knows nothing about the API, they will need to perform research to determine the ways in which they might target the application.

However, a rogue business partner or insider threat may know quite a bit about the application already without any reconnaissance. To address these distinctions, there are three basic penetration testing approaches: black box, gray box, and white box.

Black box testing models the threat of an opportunistic attacker—someone who may have stumbled across the target organization or its API. In a truly black box API engagement, the client would not disclose any information about their attack surface to the tester. You will likely start your engagement with nothing more than the name of the company that signed the SOW. From there, the testing effort will involve conducting reconnaissance using open-source intelligence (OSINT) to learn as much about the target organization as possible. You might uncover the target's attack surface by using a combination of search engine research, social media, public financial records, and DNS information to learn as much as you can about the organization's domain. The tools and techniques for this approach are covered in much more detail in Chapter 6. Once you've conducted OSINT, you should have compiled a list of target IP addresses, URLs, and API endpoints that you can present to the client for review. The client should look at your target list and then authorize testing.

A gray box test is a more informed engagement that seeks to reallocate time spent on reconnaissance and instead invest it in active testing. When performing a gray box test, you'll mimic a better-informed attacker. You will be provided information such as which targets are in and out of scope as well as access to API documentation and perhaps a basic user account. You might also be allowed to bypass certain network perimeter security controls.

Bug bounty programs often fall somewhere on the spectrum between black box and gray box testing. A bug bounty program is an engagement where a company allows hackers to test its web applications for vulnerabilities, and successful findings result in the host company providing a bounty payment to the finder. Bug bounties aren't entirely "black box" because the bounty hunter is provided with approved targets, targets that are out of scope, types of vulnerabilities that are rewarded, and allowed types of attacks. With these restrictions in place, bug bounty hunters are only limited by their own resources, so they decide how much time is spent on reconnaissance in comparison to other techniques. If you are interested in learning more about bug bounty hunting, I highly recommend Vickie Li's *Bug Bounty Bootcamp* (*https://nostarch.com/bug-bounty-bootcamp*).

In a white box approach, the client discloses as much information as possible about the inner workings of their environment. In addition to the information provided for gray box testing, this might include access to application source code, design information, the software development kit (SDK) used to develop the application, and more. White box testing models the threat of an inside attacker—someone who knows the inner workings of the organization and has access to the actual source code. The more information you are provided in a white box engagement, the more thoroughly the target will be tested.

The customer's decision to make the engagement white box, black box, or somewhere in between should be based on a threat model and threat intelligence. Using threat modeling, work with your customer to profile the organization's likeliest attacker. For example, say you're working with a small business that is politically inconsequential; it isn't part of a supply chain for a more important com-

pany and doesn't provide an essential service. In that case, it would be absurd to assume that the organization's adversary is a well-funded advanced persistent threat (APT) like a nation-state. Using the techniques of an APT against this small business would be like using a drone strike on a petty thief. Instead, to provide the client with the most value, you should use threat modeling to craft a realistic threat. In this case, the likeliest attacker might be an opportunistic, medium-skilled individual who has stumbled upon the organization's website and is likely to run only published exploits against known vulnerabilities. The testing method that fits the opportunistic attacker would be a limited black box test.

The most effective way to model a threat for a client is to conduct a survey with them. The survey will need to reveal the client's scope of exposure to attacks, their economic significance, their political involvement, whether they are involved in any supply chains, whether they offer essential services, and whether there are other potential motives for a criminal to want to attack them. You can develop your own survey or put one together from existing professional resources like MITRE ATT&CK (*https://attack.mitre.org*) or OWASP (*https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html*).

The testing method you select will determine much of the remaining scoping effort. Since black box testers are provided with very little information about scoping, the remaining scoping items are relevant for gray box and white box testing.

Which API Features You Should Test

One of the main goals of scoping an API security engagement is to discover the quantity of work you'll have to do as part of your test. As such, you must find out how many unique API endpoints, methods, versions, features, authentication and authorization mechanisms, and privilege levels you'll need to test. The magnitude of the testing can be determined through interviews with the client, a review of the relevant API documentation, and access to API collections. Once you have the requested information, you should be able to gauge how many hours it will take to effectively test the client's APIs.

API Authenticated Testing

Determine how the client wants to handle the testing of authenticated and unauthenticated users. The client may want to have you test different API users and roles to see if there are vulnerabilities present in any of the different privilege levels. The client may also want you to test a process they use for authentication and the authorization of users. When it comes to API weaknesses, many of the detrimental vulnerabilities are discovered in authentication and authorization. In a black box situation, you would need to figure out the target's authentication process and seek to become authenticated.

Web Application Firewalls

In a white box engagement, you will want to be aware of any web application firewalls (WAFs) that may be in use. A *WAF* is a common defense mechanism for web applications and APIs. A WAF is a device that controls the network traffic

that reaches the API. If a WAF has been set up properly, you will find out quickly during testing when access to the API is lost after performing a simple scan. WAFs can be great at limiting unexpected requests and stopping an API security test in its tracks. An effective WAF will detect the frequency of requests or request failures and ban your testing device.

In gray box and white box engagements, the client will likely reveal the WAF to you, at which point you will have some decisions to make. While opinions diverge on whether organizations should relax security for the sake of making testing more effective, a layered cybersecurity defense is key to effectively protecting organizations. In other words, no one should put all their eggs into the WAF basket. Given enough time, a persistent attacker could learn the boundaries of the WAF, figure out how to bypass it, or use a zero-day vulnerability that renders it irrelevant.

Ideally, the client would allow your attacking IP address to bypass the WAF or adjust their typical level of boundary security so that you can test the security controls that will be exposed to their API consumers. As discussed earlier, making plans and decisions like this is really about threat modeling. The best tests of an API will be those that align with actual threats to the API provider. To get a test that provides the most value to the security of the API, it is ideal to map out the probable adversary and their hacking techniques. Otherwise, you'll find yourself testing the effectiveness of the API provider's WAF rather than the effectiveness of their API security controls.

Mobile Application Testing

Many organizations have mobile applications that expand the attack surface. Moreover, mobile apps often rely on APIs to transmit data within the application and to supporting servers. You can test these APIs through manual code review, automated source code analysis, and dynamic analysis. *Manual* code review involves accessing the mobile application's source code and searching for potential vulnerabilities. *Automated* source code analysis is similar, except it uses automated tools to assist in the search for vulnerabilities and interesting artifacts. Finally, *dynamic* analysis is the testing of the application while it is running. Dynamic analysis includes intercepting the mobile app's client API requests and the server API responses and then attempting to find weaknesses that can be exploited.

Auditing API Documentation

An API's *documentation* is a manual that describes how to use the API and includes authentication requirements, user roles, usage examples, and API endpoint information. Good documentation is essential to the commercial success of any self-sufficient API. Without effective API documentation, businesses would have to rely on training to support their consumers. For these reasons, you can bet that your target APIs have documentation.

Yet, this documentation can be riddled with inaccuracies, outdated information, and information disclosure vulnerabilities. As an API hacker, you should search for your target's API documentation and use it to your advantage. In gray box and white box testing, an API documentation audit should be included within the

scope. A review of the documentation will improve the security of the target APIs by exposing weaknesses, including business logic flaws.

Rate Limit Testing

Rate limiting is a restriction on the number of requests an API consumer can make within a given time frame. It is enforced by an API provider's web servers, firewall, or web application firewall and serves two important purposes for API providers: it allows for the monetization of APIs and prevents the overconsumption of the provider's resources. Because rate limiting is an essential factor that allows organizations to monetize their APIs, you should include it in your scope during API engagements.

For example, a business might allow a free-tier API user to make one request per hour. Once that request is made, the consumer would be kept from making any other request for an hour. However, if the user pays this business a fee, they could make hundreds of requests per hour. Without adequate controls in place, these non-paying API consumers could find ways to skip the toll and consume as much data as often as they please.

Rate limit testing is not the same as denial of service (DoS) testing. DoS testing consists of attacks that are intended to disrupt services and make the systems and applications unavailable to users. Whereas DoS testing is meant to assess how resilient an organization's computing resources are, rate limit testing seeks to bypass restrictions that limit the quantity of requests sent within a given time frame. Attempting to bypass rate limiting will not necessarily cause a disruption

to services. Instead, bypassing rate limiting could aid in other attacks and demonstrate a weakness in an organization's method of monetizing its API.

Typically, an organization publishes its API's request limits in the API documentation. It will read something like the following:

> *You may make X requests within a Y time frame. If you exceed this limit, you will get a Z response from our web server.*

Twitter, for example, limits requests based on your authorization once you're authenticated. The first tier can make 15 requests every 15 minutes, and the next tier can make 180 requests every 15 minutes. If you exceed your request limit, you will be sent an HTTP Error 420, as shown in *Figure 0-1*.

Figure 0-1: Twitter HTTP status code from https://developer.twitter.com/en/docs

If insufficient security controls are in place to limit access to an API, the API provider will lose money from consumers cheating the system, incur additional costs due to the use of additional host resources, and find themselves vulnerable to DoS attacks.

Restrictions and Exclusions

Unless otherwise specified in penetration testing authorization documentation, you should assume that you won't be performing DoS and distributed DoS (DDoS) attacks. In my experience, being authorized to do so is pretty rare. When DoS testing is authorized, it is clearly spelled out in formal documentation. Also, with the exception of certain adversary emulation engagements, penetration testing and social engineering are typically kept as separate exercises. That being said, always check whether you can use social engineering attacks (such as phishing, vishing, and smishing) when penetration testing.

By default, no bug bounty program accepts attempts at social engineering, DoS or DDoS attacks, attacks of customers, and access of customer data. In situations where you could perform an attack against a user, programs normally suggest creating multiple accounts and, when the relevant opportunity arises, attacking your own test accounts.

Additionally, particular programs or clients may spell out known issues. Certain aspects of an API might be considered a security finding but may also be an intended convenience feature. For example, a forgot-your-password function could display a message that lets the end user know whether their email or password is incorrect; this same function could grant an attacker the ability to brute-force valid usernames and emails. The organization may have already decided to accept this risk and does not wish for you to test it.

Pay close attention to any exclusions or restrictions in the contract. When it comes to APIs, the program may allow for testing of specific sections of a given API and may restrict certain paths within an approved API. For example, a banking API provider may share resources with a third party and may not have authorization to allow testing. Thus, they may spell out that you can attack the */api/accounts* endpoint but not */api/shared/accounts*. Alternatively, the target's authentication process may be through a third party that you are not authorized to attack. You will need to pay close attention to the scope in order to perform legal authorized testing.

Security Testing Cloud APIs

Modern web applications are often hosted in the cloud. When you attack a cloud-hosted web application, you're actually attacking the physical servers of cloud providers (likely Amazon, Google, or Microsoft). Each cloud provider has its own set of penetration testing terms and services that you'll want to become familiar with. As of 2021, cloud providers have generally become friendlier toward penetration testers, and far fewer of them require authorization submissions. Still, some cloud-hosted web applications and APIs will require you to obtain penetration testing authorization, such as for an organization's Salesforce APIs.

You should always know the current requirements of the target cloud provider before attacking. The following list describes the policies of the most common providers.

Amazon Web Services (AWS) AWS has greatly improved its stance on penetration testing. As of this writing, AWS allows its customers to perform all sorts

of security testing, with the exception of DNS zone walking, DoS or DDoS attacks, simulated DoS or DDoS attacks, port flooding, protocol flooding, and request flooding. For any exceptions to this, you must email AWS and request permission to conduct testing. If you are requesting an exception, make sure to include your testing dates, any accounts and assets involved, your phone number, and a description of your proposed attack.

Google Cloud Platform (GCP) Google simply states that you do not need to request permission or notify the company to perform penetration testing. However, Google also states that you must remain compliant with its acceptable use policy (AUP) and terms of service (TOS) and stay within your authorized scope. The AUP and TOS prohibit illegal actions, phishing, spam, distributing malicious or destructive files (such as viruses, worms, and Trojan horses), and interruption to GCP services.

Microsoft Azure Microsoft takes the hacker-friendly approach and does not require you to notify the company before testing. In addition, it has a "Penetration Testing Rules of Engagement" page that spells out exactly what sort of penetration testing is permitted (*https://www.microsoft.com/en-us/msrc/pentest-rules-of-engagement*).

At least for now, cloud providers are taking a favorable stance toward penetration testing activities. As long as you stay up-to-date with the provider's terms, you should be operating legally if you only test targets you are authorized to hack and avoid attacks that could cause an interruption to services.

DoS Testing

I mentioned that DoS attacks are often off the table. Work with the client to understand their risk appetite for the given engagement. You should treat DOS testing as an opt-in service for clients who want to test the performance and reliability of their infrastructure. Otherwise, work with the customer to see what they're willing to allow.

DoS attacks represent a huge threat against the security of APIs. An intentional or accidental DoS attack will disrupt the services provided by the target organization, making the API or web application inaccessible. An unplanned business interruption like this is usually a triggering factor for an organization to pursue legal recourse. Therefore, be careful to perform only the testing that you are authorized to perform!

Ultimately, whether a client accepts DoS testing as part of the scope depends on the organization's *risk appetite*, or the amount of risk an organization is willing to take on to achieve its purpose. Understanding an organization's risk appetite can help you tailor your testing. If an organization is cutting-edge and has a lot of confidence in its security, it may have a big appetite for risk. An engagement tailored to a large appetite for risk would involve connecting to every feature and running all the exploits you want. On the opposite side of the spectrum are the very risk-averse organizations. Engagements for these organizations will be like walking on eggshells. This sort of engagement will have many details in the scope: any machine you are able to attack will be spelled out, and you may need to ask permission before running certain exploits.

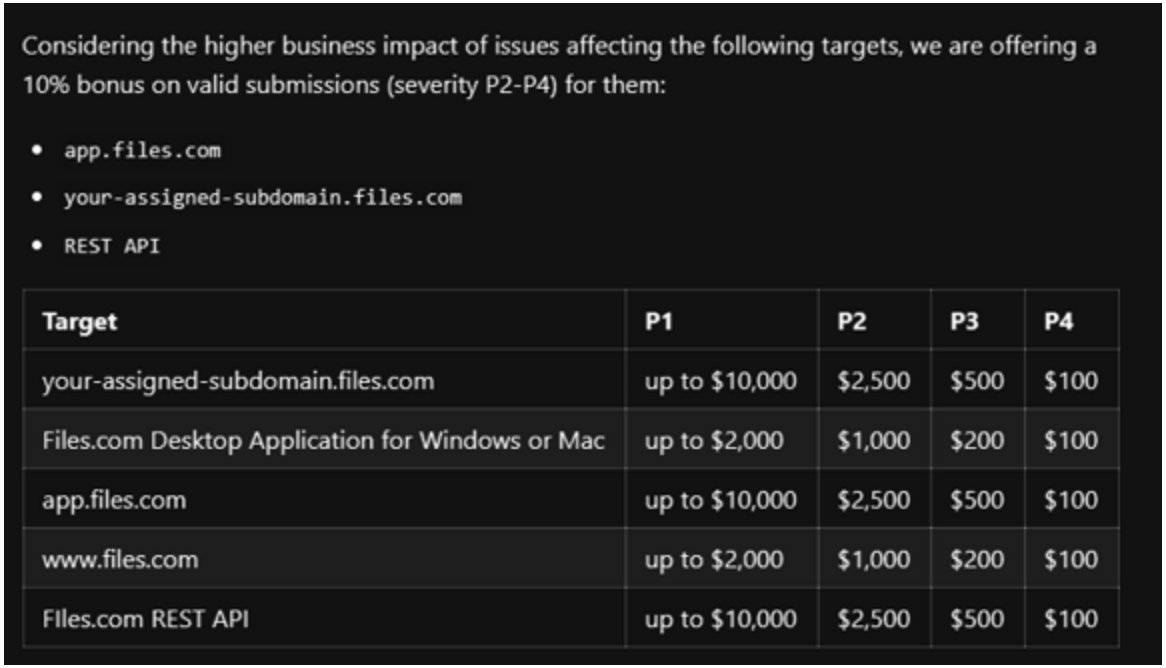
Reporting and Remediation Testing

To your client, the most valuable aspect of your testing is the report you submit to communicate your findings about the effectiveness of their API security controls. The report should spell out the vulnerabilities you discovered during your testing and explain to the client how they can perform remediation to improve the security of their APIs.

The final thing to check when scoping is whether the API provider would like remediation testing. Once the client has their report, they should attempt to fix their API vulnerabilities. Performing a retest of the previous findings will validate that the vulnerabilities were successfully remediated. Retesting could probe exclusively the weak spots, or it could be a full retest to see if any changes applied to the API introduced new weaknesses.

A Note on Bug Bounty Scope

If you hope to hack professionally, a great way to get your foot in the door is to become a bug bounty hunter. Organizations like BugCrowd and HackerOne have created platforms that make it easy for anyone to make an account and start hunting. In addition, many organizations run their own bug bounty programs, including Google, Microsoft, Apple, Twitter, and GitHub. These programs include plenty of API bug bounties, many of which have additional incentives. For example, the Files.com bug bounty program hosted on BugCrowd includes API-specific bounties, as shown in *Figure 0-2*.



Considering the higher business impact of issues affecting the following targets, we are offering a 10% bonus on valid submissions (severity P2-P4) for them:

- app.files.com
- your-assigned-subdomain.files.com
- REST API

Target	P1	P2	P3	P4
your-assigned-subdomain.files.com	up to $10,000	$2,500	$500	$100
Files.com Desktop Application for Windows or Mac	up to $2,000	$1,000	$200	$100
app.files.com	up to $10,000	$2,500	$500	$100
www.files.com	up to $2,000	$1,000	$200	$100
Files.com REST API	up to $10,000	$2,500	$500	$100

Figure 0-2: The Files.com bug bounty program on BugCrowd, one of many to incentivize API-related findings

In bug bounty programs, you should pay attention to two contracts: the terms of service for the bug bounty provider and the scope of the program. Violating either of these contracts could result not only in getting banned from the bug bounty provider but legal trouble as well. The bounty provider's terms of service will contain important information about earning bounties, reporting findings, and the relationship between the bounty provider, testers, researchers, and hackers who participate and the target.

The scope will equip you with the target APIs, descriptions, reward amounts, rules of engagement, reporting requirements, and restrictions. For API bug bounties, the scope will often include the API documentation or a link to the docs. *Table*

0-1 lists some of the primary bug bounty considerations you should understand before testing.

Table 0-1: *Bug Bounty Testing Considerations*

Targets	URLs that are approved for testing and rewards. Pay attention to the subdomains listed, as some may be out of scope.
Disclosure terms	The rules regarding your ability to publish your findings.
Exclusions	URLs that are excluded from testing and rewards.
Testing restrictions	Restrictions on the types of vulnerabilities the organization will reward. Often, you must be able to prove that your finding can be leveraged in a real-world attack by providing evidence of exploitation.
Legal	Additional government regulations and laws that apply due to the organization's, customers', and data center's locations.

If you are new to bug hunting, I recommend checking out BugCrowd University, which has an introduction video and page dedicated to API security testing by

Sadako (*https://www.bugcrowd.com/resources/webinars/api-security-testing-for-hackers*). Also, check out *Bug Bounty Bootcamp* (No Starch Press, 2021), which is one of the best resources out there to get you started in bug bounties. It even has a chapter on API hacking!

Make sure you understand the potential rewards, if any, of each type of vulnerability before you spend time and effort on it. For example, I've seen bug bounties claimed for a valid exploitation of rate limiting that the bug bounty host considered spam. Review past disclosure submissions to see if the organization was combative or unwilling to pay out for what seemed like valid submissions. In addition, focus on the successful submissions that received bounties. What type of evidence did the bug hunter provide, and how did they report their finding in a way that made it easy for the organization to see the bug as valid?

Summary

In this chapter, I reviewed the components of the API security testing scope. Developing the scope of an API engagement should help you understand the method of testing to deploy as well as the magnitude of the engagement. You should also reach an understanding of what can and can't be tested as well as what tools and techniques will be used in the engagement. If the testing aspects have been clearly spelled out and you test within those specifications, you'll be set up for a successful API security testing engagement.

In the next chapter, I will cover the web application functionality you will need to understand in order to know how web APIs work. If you already understand web

application basics, move on to Chapter 2, where I cover the technical anatomy of APIs.